Trajectory Tracking of a Statically-stable Biped with Two Degrees of Freedom

By

Joseph Ewell Trout

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

IN

MECHANICAL ENGINEERING

October 3, 2003

Blacksburg, Virginia

Dr. Will Saunders, Co-chair

Dr. Pushkin Kachroo, Co-chair

Dr. Don Leo

Keywords: Autonomous, Control, Static Stability, Biped, Robotic

Trajectory Tracking of a Statically-stable Biped with Two Degrees of Freedom

Joseph Ewell Trout

Abstract

This research investigates the possibility of controlling a simple biped having two degrees of freedom only. The biped robot walked on large feet. Having large feet enabled the robot to stand on one leg stably. At any time, the robot's center of gravity remained above the area covered by one of the feet. Two servos actuated the two degrees of freedom tilting the robot to the side or moving the legs forward and backward. The biped moved by alternately tilting and striding. Turns were produced by dragging the feet along the ground. As the feet dragged, the friction generated under the feet created a turning moment that rotated the robot. Thus, the robot was able to step and turn on a flat surface.

A control algorithm was developed to attempt trajectory tracking with the biped. Trajectories along a surface can be defined in terms of linear and angular velocities. In this research, it was assumed that a high level controller had transformed a desired trajectory into discrete steps of linear and angular velocities. Motion tests showed how various settings of the servos affected the step length and turning angle of the robot. To produce the desired velocities, a program was created to select the servo commands and set the speed parameters. This program applied knowledge of the expected step length and turning angle and performed feedforward control of the velocities. This investigation identified a trajectory tracking scheme that could be used in an observer feedback scenario to achieve accurate control.

Acknowledgements

I want to offer my thanks to Dr. Saunders, my advisor in Mechanical Engineering. His Mechatronics class began my exposure to all the main ideas in this thesis.

I would like to thank Dr. Kachroo for giving me confidence to keep going and keep learning. His enthusiasm has a way of finding a next step past every dead end.

I cannot express the gratitude that my beautiful wife Jessica deserves - for taking care of everything for the past several weeks while I worked on this, for encouraging me, for endless patience, for her sweet heart smoothing over the rough edges that working on a thesis can expose.

Most importantly, I thank God for everything.

Table of contents

List of figures

List of tables

Chapter 1 - Introduction

1.1 - Motivation

For several years, the author of this paper has studied the problem of land mine detection and removal. Autonomous robots may provide the safest and most dependable solution to this problem. Currently, the rate at which land mines are planted exceeds the rate at which planted mines are removed. Even with the present demining efforts, the number of mines that threaten human lives continues to increase. To make matters worse, many of the world's concentrations of land mines are in developing countries with low socioeconomic status. People in these countries, in desperate need of a livelihood, are forced to cultivate and live on mine fields. People still search for mines with knives and excavate the mines by hand. Instead, mobile robots could be used to perform the essential task of demining [1].

Walking robots would suit the outdoor environments where demining takes place. Rugged terrain can be impassible to wheeled vehicles. On the other hand, legged mechanisms conceivably could be made to walk over any type of terrain. Mountain goats provide an excellent example of the capabilities that legged locomotion can provide. Analyzing or mimicking the gait of mountain goats might be a complex project. However, a robot mimicking a mountain goat could be an effective platform for removing mines from rugged landscapes.

To make the problem of walking robots more accessible, work can be restricted to studying statically-stable walkers. The author experimented with statically-stable walking by designing and constructing a statically-stable biped. Plans for constructing this robot are included in the appendix. The robot was constructed using the Lego Mindstorms Invention

System 2.0. By tilting a weight from side to side, the center of gravity transferred from one foot to the other. The legs were actuated lift and move forward periodically. When the weight tilted over one foot, the other foot moved forward. This robot had several problems including loose hip joints and an unreliable control system. Another limitation to this robot was that it could only walk forward in a straight line.

The biped featured in this report was similar to the Lego® robot. Instead of using Legos®, this robot was constructed of balsa wood. Rather than tilting a weight to shift the center of gravity, this robot tilted its whole body. The robot was able to turn by dragging its feet in opposite directions on the ground. Also, the control system on the robot enabled it to follow a path at set velocities.

To extend the significance of this research, a similar biped could be constructed for outdoor use. A considerable advantage to the design of the robot in this research was the simplistic design. Constructing the robot required few tools, and making repairs was easy. For field applications like demining, an easily repaired robot would be especially beneficial. Deminers in developing countries could be trained in repairing the robot. Most likely the deminers would not have experience in robotics. However, a robot like the one considered in this research would be more accessible to new users and require less background expertise. Rather than using balsa wood, the robot could be constructed with weather-proof materials. Telescoping legs could be added to enable the robot to step over bumps in the terrain. By receiving positional feedback from beacons, the robot could be controlled to follow a path through a minefield. In conclusion, the principles in this research could serve as the basis for a demining robot in the future.

1.2 - Related research

The study of walking robots has attracted a great deal of research. Most research focuses on smooth coordination of joints and linkages to make a robot walk stably over a horizontal, even surface. Raibert pioneered research on dynamically-stable legged robots [2]. The term dynamically-stable refers to a system that must keep moving in order to maintain stability. Legged organisms often rely on dynamic stability to balance themselves. Humans, for example, walk with a dynamically-stable gait. A person repeatedly leans forward from one leg and falls onto the other leg. During the falling period of the walk, a person cannot stop instantly and remain stable. Raibert began by studying one-legged locomotion and constructed a robot resembling a pogostick. A boom extending from the robot prevented it from tipping sideways. This monopod moved by controlling its hopping height and the placement of its foot ahead of its body. After successfully developing a monopod robot, Raibert extended his breakthroughs to bipedal and quadrupedal robots. All of Raibert's robots performed dynamically-stable motions including a variety of gaits and maneuvers. Although these robots could travel at high speeds over smooth or slightly irregular terrain, they could not traverse the irregularities of natural terrain.

Since Raibert began his work with walking robots, many other researchers have developed their own walkers. The vast majority of the research is limited to smooth terrain. A few studies have focused on adapting to changes in terrain [3], [4], [5], [6], [7], [8], [9]. Still, the type of terrain that this research enabled robots to traverse had only slight irregularities or consisted of steps. A robot that can maneuver effectively in natural environments remains to be

developed. In an effort to improve the abilities of legged robots, most of the robot designs have grown increasingly complex.

On the other hand, some researchers have sought to minimize the number of actuators used in their walkers. McGeer initiated the study of "passive" walkers [10], [11]. These mechanisms walk downhill without any actuation or active control system. As the walkers move downhill, the conversion of gravitational potential energy to kinetic energy provides all the power for locomotion. The mechanics of a passive walker create a periodic motion, in which the legs alternately step and swing. Research in passive walking has lent insight into the nature of human locomotion. The process of human walking involves a naturally balanced transition from one freely swinging leg to the other. Humans exert energy to push off with their toes, to lock their knees, and to cushion their footfalls. However, the rest of human walking follows the passive walking patterns.

Although McGeer's walkers were prevented from tilting to the side, other studies have considered three-dimensional passive walking. Coleman and Ruina constructed a walker out of Tinkertoys® [12], [13]. Their walker wobbled side to side and walked downhill. When the walker tilted to one side, the leg on the opposite side swung forward. This swinging leg caused the walker to tilt in the opposite direction. The stance leg then lifted and swung. Thus, the cycle of tilting and swinging continued as the walker progressed downhill. By including tilting motions, Art Kuo expanded the theories of passive walking into three dimensions [14]. Kuo analyzed a three-dimensional passive walker. Several of these type walkers have been designed and tested [15], [16]. However, the precision required to generate periodic motions has made three-dimensional passive walking extremely difficult to achieve. Even slight errors in the initial positioning and starting velocities of the walkers will propagate until the walkers collapse.

Research has shown that minimal control can enhance the performance of otherwise passive walkers. By controlling the orientation of the ankle joints, a passive walker can be stabilized [17]. Other work has considered the potential of adding actuation to passive walkers [18], [19], [20]. By using the passive walking motions, efficiency can be increased. Actuation could be added at certain points during walking. Then the passive motion would take over for the rest of the cycle.

To avoid the complexity that dynamic stability adds, robotic walkers could be designed to maintain static stability. While walking, the robot's center of gravity would remain above the area supported by the feet. By keeping at least three feet in contact with the ground, a stable tripod would support the body at all times. A quadruped could achieve static stability by moving one leg at a time. Hexapods could alternate moving sets of three legs. Another approach would be to incorporate as many legs as possible and produce a caterpillar-like gait [21].

Even in the field of statically-stable walking, bipeds remain a subject of interest [3], [4], [5]. To maintain static stability, a biped's center of gravity must stay above the region covered by the stance foot. If the center of gravity moves outside the vertical projection of the stance foot, the biped will fall. Feet that cover a greater area provide a larger range of tilting before the robot will lose stability.

The research in this paper investigates control of a statically-stable biped. The control system used with this walker did not consider the robot's dynamics. There was no need to dynamically stabilize the walker. At all times, the center of gravity remained above the area covered by the robot's large feet. A particularly novel aspect of this research was the elucidation of the performance gained by actuating two degrees of freedom only. Controlling only two

degrees of freedom enabled the biped to achieve trajectory tracking. Therefore, this research demonstrates seeks the advantage of legged mobility while still achieving simplicity.

1.3 - Contribution

The robot studied in this research is the first statically-stable two degree-of-freedom biped proposed for trajectory tracking. While the current trend in robotics tends to produce increasingly complex designs, this robot maintains simplicity. A simple design has several advantages. It would be easier for unskilled operators to maintain this biped than a more complicated robot. Repairs could be made quickly with few tools. In case the robot was destroyed, the loss would be less costly, and a replacement would be less expensive.

A scenario can be imagined in which this robot would perform demining. In this research, the robot walked on a flat surface using feedforward control. Natural terrain, in which land mines are buried, would include contours and irregularities. Redesigning the robot with longer, extendable legs could provide terrain adaptability. Stronger, more durable materials could make the robot sturdy and resistant to harsh weather.

Although the robot used in this research exhibited significant tracking error, a feedback system could correct for this error. To provide feedback, a camera could be mounted where its field of vision would include the robot. Image processing would locate a certain feature on the robot and track the position of this feature over the terrain. An observer function could infer the linear and angular velocities of the robot based on changes in the robot position. Negative feedback of the error in these measured velocities could be used to adjust the reference trajectory input to the controller. Thus, the velocities for the next step could be changed to correct for the

tracking error. In this scenario, the feedforward controller designed in this research could be used in concert with the feedback to produce accurate trajectory tracking.

Chapter 2 - Robot construction

2.1 - Overview of the design

A simple walking mechanism served as the robot platform for this research. Using two legs, which were actuated by two servo motors only, the robot could waddle around flat, even terrain. This simple walker constructed with minimal components provided an inexpensive, durable platform, which was ideal for experimental research. A CAD rendering of the fully assembled robot is shown in Figure 2-1. Assembling the robot required common tools and was completed in a few hours. Because the robot design constrained all motions to two degrees of freedom, even the complexity of the kinematic analysis was limited.

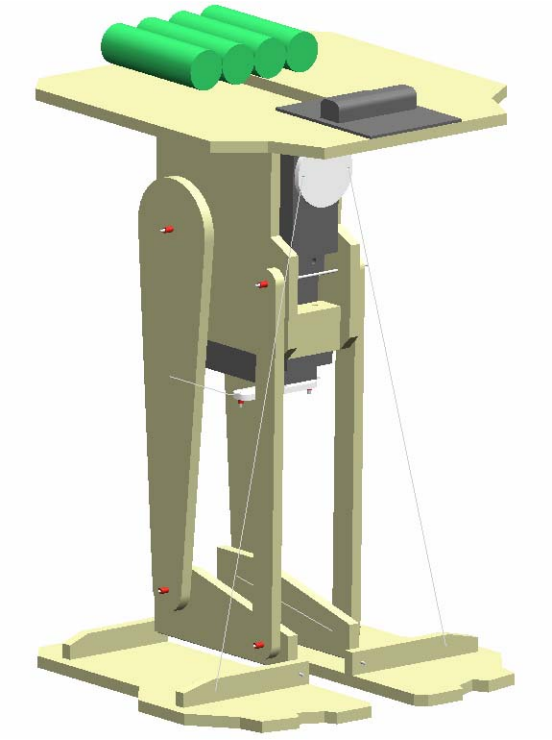


Figure 2-1. Fully assembled robot used in the experiment. The view shows the front and right side of the robot as it stands upright. This bipedal walker was powered by four AA batteries (represented by the row of cylinders), controlled by a Stamp I microcontroller (represented by the black board on top) and actuated by two servo motors (mounted within the torso). Wires are shown connecting the white servo arms to the feet and legs.

This walker was purchased as the Bigfoot robot kit from Milford Instruments. The Bigfoot robot is a two-legged robot driven by two servo motors only. Operating according to its original design, Bigfoot walks forward until bumping into an obstacle with a touch sensor. Then the robot walks in reverse, turns to the side, and resumes forward walking. The simplistic design of a bipedal walker actuated by two servos was adopted in this research. The components were assembled largely as indicated by Bigfoot's assembly instructions to build the same mechanism as intended from the kit. However, several modifications were made to the construction. Also Bigfoot's touch sensors were omitted from the robot construction, because this research considered feedforward control exclusively.

The robot body consisted of relatively few components. Most of the structure was created by joining pieces of balsa wood with wire connectors. All of the balsa wood members were precut and included in the Bigfoot kit. The wire connectors had to be cut and shaped using pliers as directed in the assembly instructions. Four AA Nickel-metal Hydride batteries supplied power to the robot. Compared to Alkaline batteries, Nickel-metal Hydride batteries are better for supporting the high current draws that would be produced by the servos. Each servo motor actuating the robot was a Hitec model HS-311. Mounted on the front of the torso, the roll servo actuated the tilt of the body. The pace servo mounted under the torso and actuated the spreading of the legs forward and backward. Wired between the power supply and servos, a Basic Stamp I microcontroller commanded the servo motions.

2.2 - Assembly of the robot

As shown in Figure 2-2, each foot consisted of three pieces of wood. The large pad of the foot provided a stable platform to help maintain static stability. Affixed to the top surface of the pad, two strips of wood held the points by which the foot connected to the robot body.

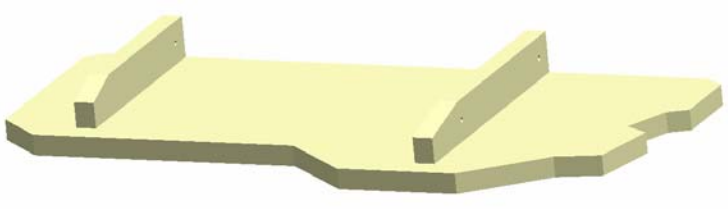


Figure 2-2. Rendering of the foot including the balsa wood strips adhered to the top surface.

Above the foot, a piece of balsa wood served as the connecting point for the ankle joints. Figure 2-3 shows the ankle block connected to the foot. A wire, which stretched between the wooden strips on the foot, was mounted with Super Glue to the bottom of the ankle block. Although a hot glue gun was recommended in the assembly instructions, Super Glue was substituted in the bond under the ankle block. However, the bond repeatedly peeled away from the ankle block during normal wear and tear of the robot. In addition to the glue bond, wrapping masking tape around the ankle block finally provided enough strength to hold the parts together. Another piece of wire was wrapped through two holes in the ankle block to form the joints with the leg pieces.

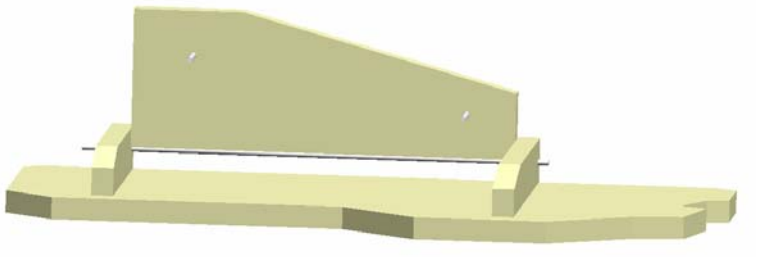


Figure 2-3. Assembly model of the ankle block and foot. The ankle block is shown glued onto the wire that protrudes through the wood strips on the foot.

Each leg consisted of two parallel links connecting the ankle block and the torso. Both links attached to the ankle by the wire joints described above. At the opposite end of the links, two wires inserted through the torso held the links to the torso. The parallel links with equal length created a parallelogram between the hip and ankle joints. Figure 2-4 highlights the parallelogram formed between the leg joints. Because the joints always remained in a parallelogram configuration, the kinematic analysis was simplified. The parallel linkage formed with the legs also assured that the bottom of the ankle block remained parallel to the ground even when the foot was lifted. Thus, the orientation of the ankle block and foot prevented the robot from tipping in the pitch direction.

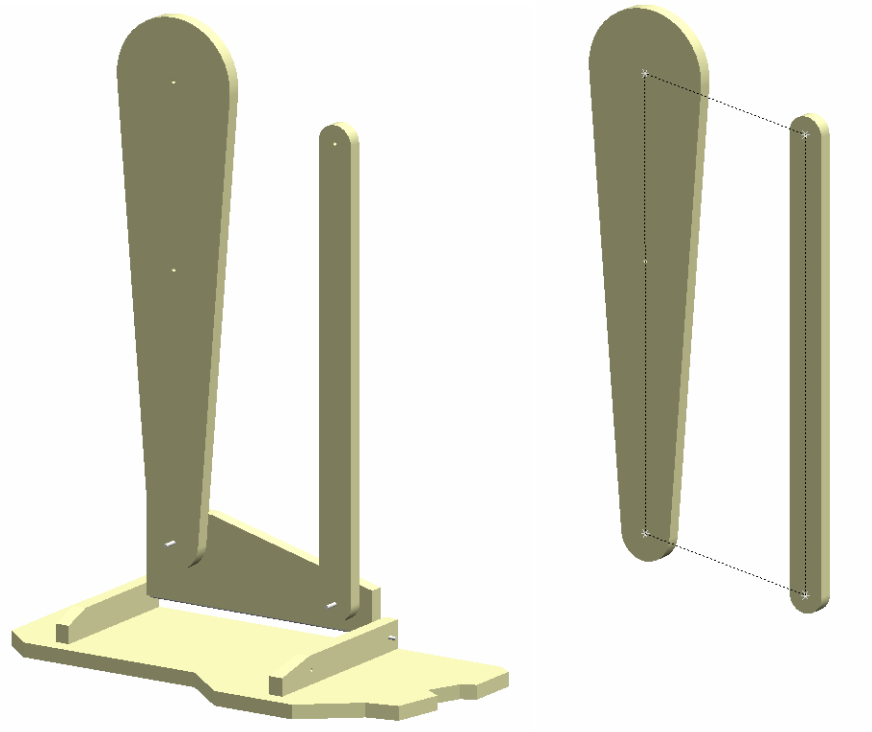


Figure 2-4. Assembly of the leg links attached to the ankle block shown to the left. On the right, the parallelogram formed by the leg joints is highlighted.

Constructing the torso involved several changes from Bigfoot's instructions. The instructions suggested that a hot glue gun be used to mount the torso panels directly to the sides of the servo motors. Rather than affixing the servos permanently to the torso panels, an alternative was desired to allow easy removal of the servos in case adjustments later became necessary. The use of screws instead of glue left the servos removable. The servos were molded with mounting slots for screws, and the screws were included in the servo packages. The mounting slots were located such that the screws would insert parallel to the torso panels instead of screwing into the face of the panels. Gluing small blocks of wood to the torso panels provided a structure that could hold the screws. In Figure 2-5 the blocks are shown glued to the one of the torso panels. Then the servos could be screwed into these blocks of wood, and the servo wires routed through the rear of the torso. When the blocks of wood had been glued in place, there was enough space for each servo casing to fit between the blocks. However, sufficient clearance was not available for the wires protruding from the servo casing. The wires interfered with the wooden mounting blocks and the servos would not slide into place. Finally, notches cut into the mounting blocks allowed the servos to be inserted and attached to the torso.

At certain wire joints, rubber tubing was used to hold the joint together. The rubber tubing fit snuggly around the wire and resisted slipping. An example of the way in which the rubber tubing was attached is depicted in Figure 2-6. One piece of the tubing was cut into twelve segments to hold the joints as recommended in the assembly instructions. According to the instructions, the tubing should be applied to the ankle joints, hip joints, and the joints connecting the pace servo to the wider leg links. No tubing was recommended for the joints connecting the roll servo to the feet. Eventually, small segments of tubing were applied to these joints. The only joints not requiring tubing were the joints between the ankle blocks and feet. Constrained

between the two wood strips on the foot, the ankle block could not slide along the axis of the wire joints. Therefore, tubing was unnecessary to prevent the joints from loosening or separating.

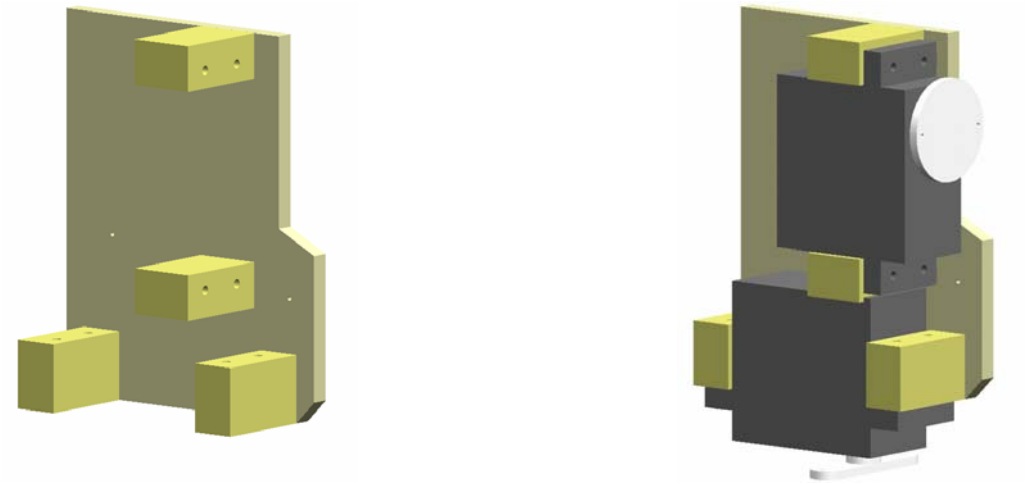


Figure 2-5. Two stages of the torso construction showing the small blocks glued in place in the image to the left and the servos inserted between the blocks in the image to the right.

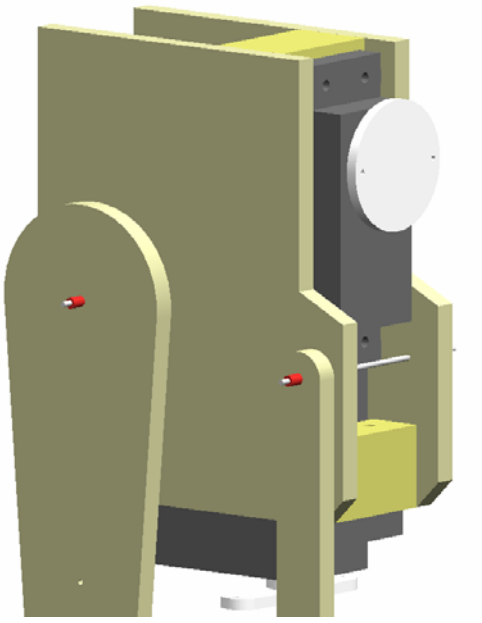


Figure 2-6. Torso assembled and the wires forming the hip joints inserted through the torso. The rings made of rubber tubing are shown wrapped around the wires and pressed against the leg links. Fitting tightly around the wires, these rubber rings were intended to keep the joints tight and prevent the legs from sliding off the wire ends.

Although the rubber tubing was intended to provide all necessary support to the joints, many joints remained loose. During experimentation, the wires connecting the roll servo to the feet experienced excessive slipping and even disconnected. Also the joints at the hips and ankles tended to loosen slowly. The friction between the tubing and wire did not prevent the tubing from gradually sliding away from the joints. At the ankle joints especially, the tubing could not hold the joints tight. Even when the segments of tubing were pressed tightly against the ankle joints, the leg links wobbled at the ankle joints. The joints that experienced the least loosening were the joints connecting the pace servo to the legs. These joints upheld best because the bends in the wire reduced the stress on the tubing. The ends of the wire hooked slightly around the servo arm and the holes in the legs. Therefore, the hooks in the wire helped to prevent the wire from sliding out of place.

To tighten the joints, modifications were made to the original design. Because the joints that linked to the pace servo remained tight, half of the rubber tubing from these joints was borrowed for less secure joints. Cutting each segment of tubing in half provided four extra segments. These four half segments were added to the joints between the pace servo and feet. On all joints, Super Glue was applied at the ends of the tubing to bond the tubing and wire together. Gluing strengthened the hold that the tubing had on the wire. A stronger hold resisted the tubing from sliding away from the joints. After the glue was added, the joints proved to be less prone to loosen. Still, the hip and ankle joints occasionally loosened. After these joints loosened, the tubing had to be reset with glue. Periodically the tubing was removed from the loosened joints. Then the tubing and wire were cleaned and glued again.

Even with glue added, the ankle joints exhibited looseness and wobbling. The looseness of the ankles actually helped the robot to perform as desired. When the robot tilted to one side, the opposite foot would be expected to rise if the joints held tightly. However, when the robot tilted slightly, both feet remained on the ground. The ankle joints appeared to be the main cause of this lack of response. After tilting the robot to one side, I could cause the opposite foot to lift simply by holding the ankle joints tighter with my fingers. The play in the ankle joints aided the performance of the robot, because turning motions required that both feet touch the ground. Frictional forces between the feet and the ground generated the turning moments. The loose ankle joints allowed both feet to touch the ground during small tilt angles. Varying the tilt angle could change the distribution of the robot's weight between the feet rather than lifting one foot entirely. Changes in the weight distribution affected the turning angle and the distance that the feet slipped along the ground. Thus, setting the tilt angle controlled the robot's turning angle and step length. On the other hand, the looseness of the ankle joints was not measured and may have varied unpredictably. These unknown effects of the ankle joints may have introduced error in the turning angle.

Additionally, the wire attached to the bottom of the ankle block did not remain firmly attached on the right foot. Part of the glue bond separated between the wire and the ankle block. Although the masking tape still held the wire, the rear of the ankle block could pull away from the wire by about one millimeter. This play at the rear of the right ankle was not obvious while the robot walked. However, the looseness did contribute to the left foot dragging on the ground while the robot tilted right. Like the ankle joints, this loose wire contributed to the control of the turning angle and step length.

Although the construction of the robot raised several concerns, the robot was retained as the platform for this research. The pieces that occasionally disconnected were checked before running every test in the experiments. Whenever parts were not firmly connected, repairs were made. The loose joints were tightened and reglued periodically. Because the maintenance was easy to perform, the problems with the robot were not detrimental. However, the unpredictable looseness of the joints may have contributed to the error in predicting the robot movements. Distances that the robot stepped or angles that it turned were predicted with a large range of error. The robot was controlled to produce the same trends as a desired `path, but accumulated error caused the robot to diverge from the path as time progressed. In this research, the interest was in demonstrating the potential of controlling this type of two-degree-of-freedom robot to follow a trajectory. For future research, feedback control could correct for the error encountered with this robot platform. Also, redesigning the robot with metal or plastic construction and secure bearings at each joint could improve the repeatability of the movements. A more durable construction would make the robot better suited for outdoor applications.

Chapter 3 - Kinematic analysis

Performing kinematic analysis provided a complete description of the robot body positions. Every joint location was defined based on the position of the servo motors. Thus, for any position of the servos, the coordinates of each joint in three-dimensional space could be calculated. Typically, kinematic analysis is necessary as a step to solving the dynamic equations of motion for a mechanism. For this research, the kinematic analysis was used in simulating the expected motion of the robot. A stick figure of the robot was animated based on the kinematic equations. Also, the expected location of each foot was used in evaluating the final performance of the robot. This chapter will discuss the kinematic analysis used in solving revolute-spheric-spheric-revolute linkages connected to each servo. Then, the resulting position of each joint will be related to global coordinates.

Based on the motions of the servo motors, the positions of all joints on the robot could be found. A kinematic analysis of the robot was conducted to determine these joint locations. Figure 3-1 and Figure 3-2 show the notation used in naming the joints. Wires connected the roll servo motor to the outstep of each foot. When the roll servo turned to lift on the outstep of one foot, the servo pressed down on the other servo. This opposed lifting and pressing down caused the robot to tilt to the side being lifted. The pace servo connected to the legs and actuated the striding motion. While the pace servo pulled one leg forward, the other leg swung backward. First, kinematic equations were solved to relate the servo positions to the tilt and stride angles. Then, the relationship of the tilt and stride angles to the joint locations in three-dimensional space were solved.

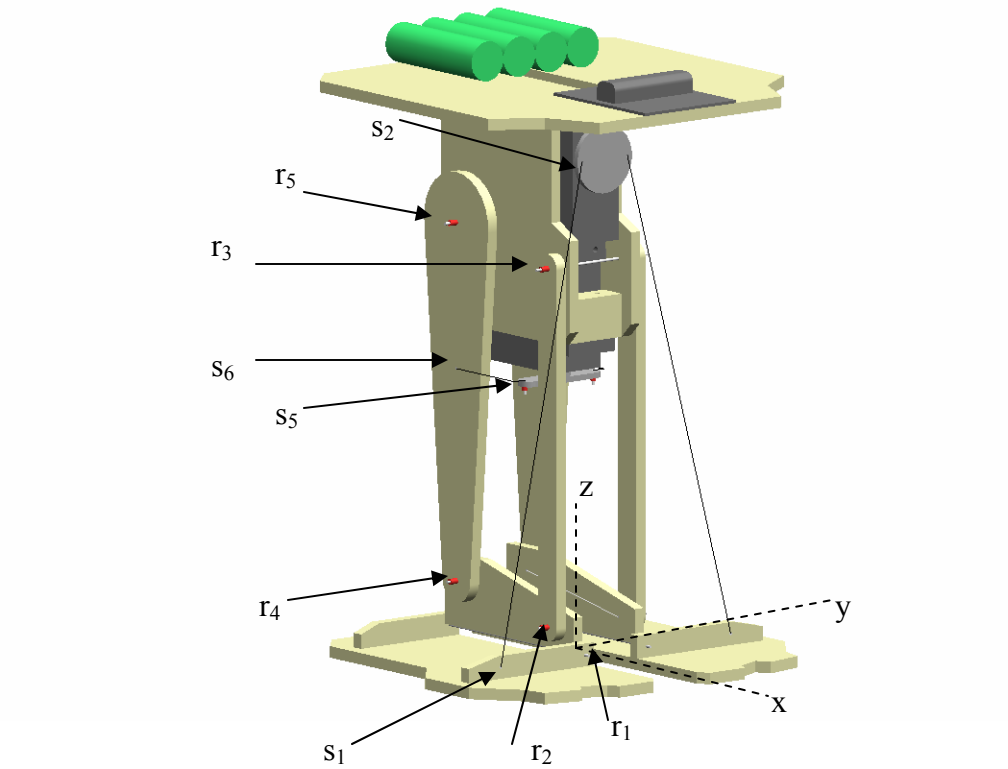


Figure 3-1. Image showing the notation for the joints on the right side of the robot.

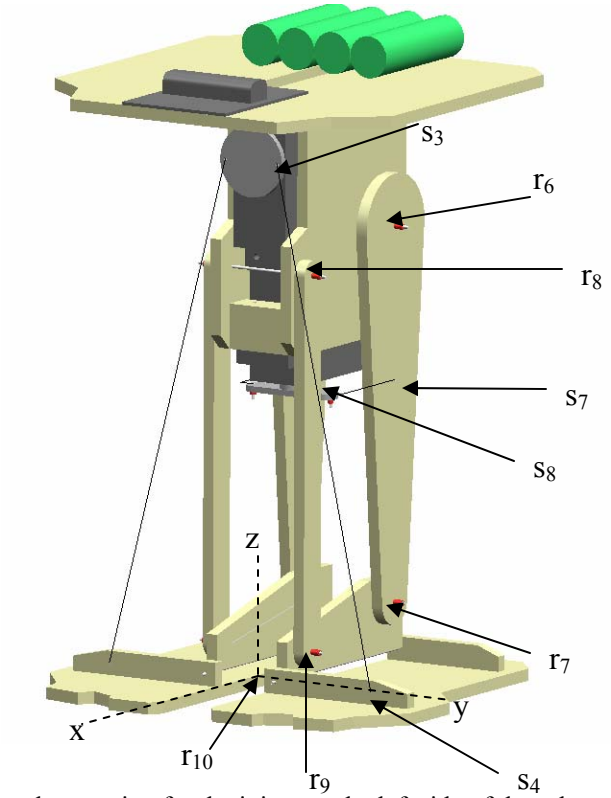


Figure 3-2. Image showing the notation for the joints on the left side of the robot.

3.1 - RSSR analysis

Relating the servo positions to the tilt and stride angles required the solution of revolute-spheric-spheric-revolute (RSSR) linkages. Each of the linkages formed between the servos and the robot's body contained two revolute joints connected with two spheric joints. Thus, the servos affected the motion of the robot through revolute-spheric-spheric-revolute linkages. A revolute joint allows only rotation about the joint axis. A spheric joint permits rotation about three perpendicular axes. To provide examples, human knee joints are revolute and shoulder joints are spheric.

Analysis of the RSSR linkages with the pace servo was performed by considering the right side of the robot. The parts involved in the linkage are shown in Figure 3-3. On the robot's left side, the linkage was a mirror image of the right side configuration. From the leg, a wire led to the servo arm. Both joints formed between the wire and the leg or servo arm functioned as spheric joints. The leg rotated about the revolute joint at the hip. Also, the center of the servo acted as a revolute joint.

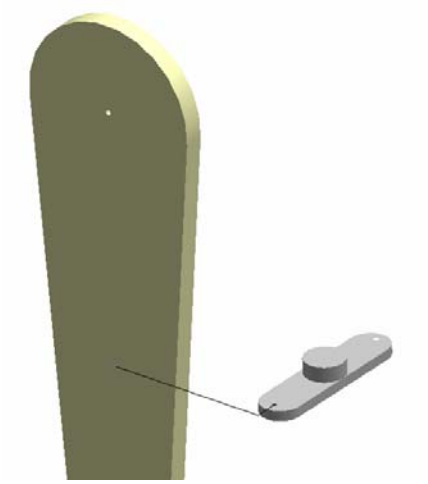


Figure 3-3. Pictured at the left, the leg link connected to the pace servo arm shown to the right. The black line between the two parts depicts the wire, which joined the leg and servo arm.

To analyze the RSSR linkage, the location or orientation of the joints had to be known. A symbolic representation of the RSSR linkage on the right side of the robot appears in Figure 3-4. The revolute joints at the pace servo, r_p, and the hip joint, r_5, connect to the wire at the spheric joints, s_5 and s_6. An arbitrary origin was created at the center of r_p with the z-axis and joint axis aligned. Dimensions were defined relative to the origin with zero rotation angle at the pace servo and hip joint. The servo arm connecting r_p and s_5 had a length of 15 mm. The length of the wire was 29 mm, and the distance between the hip joint and the point where the leg connected to the wire was 51 mm. Given a rotation of the pace servo the position of s_5 could be found by

$$s_5 = \begin{bmatrix} 15mm\sin\theta_p & -15mm\cos\theta_p & 0 \end{bmatrix} \tag{3.1}$$

where θ_p is the angle of rotation of the pace servo. Similarly, the position of s_6 was determined by

$$s_6 = \begin{bmatrix} -29 + 51mm\sin\theta_s & -15 & 51 - 51mm\cos\theta_s \end{bmatrix} \tag{3.2}$$

where θ_s is the angle of rotation of the hip joint, or the stride angle.

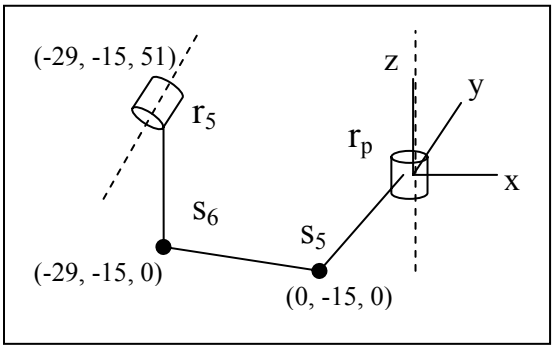


Figure 3-4. Symbolic representation of the RSSR linkage joining the pace servo to the right leg. The dimensions are shown for the initial configuration with zero rotation in all joints.

Solving the relationship between the servo and leg involved an iterative technique. A list of pace servo angles was compiled from which the corresponding hip angles were solved. The affects of the revolute joints on the motions of the linkage are illustrated in Figure 3-5. For each angle of the pace servo, the coordinates of s_5 were calculated. Then the angle of the hip joint was adjusted until the location of s_6 was 29 mm from s_5. This distance indicated that the spheric joints were separated by the length of the wire. Thus, the linkage could be put together with the wire connecting the servo arm to the leg link. The angle of r_5 that connected the linkage was the angle to which the servo would drive the leg. No fast convergence techniques were used to solve the angle of r_5. A brute force method was used. The angle of r_5 was changed by one ten-thousandth of a radian in the same direction of the angle of r_p. If the linkage would not connect, the angle of r_5 was changed again until reaching the solution. The solutions placed the spheric joints at a distance of 29 mm within one hundredth of a millimeter. The measurements used in the calculations were accurate only to one millimeter, so the precision of the RSSR solution was limited more by the measurements than the calculations.

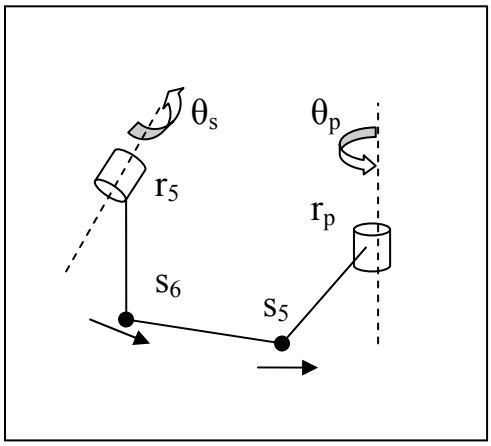


Figure 3-5. Illustration of the interaction between the motion of the spheric joints and the angular positions of the revolute joints.

Analysis of the RSSR linkages connecting the pace servo to the robot legs was performed independently from considering the roll position. Later, the analysis of the roll linkages needed to include the position of the pace servo. Because the servo arm connected directly to the leg link via the wire, the servo controlled the stride angle of the leg directly. In the roll linkages, however, the stride angle influenced the relationship between the roll servo and the tilt angle. The stride angle determined the position of the revolute joint on the foot link. Thus, the pace position affected the dimensions used in the RSSR analysis of the roll linkage. For each solution of the pace linkage, the angle of r_5 was incorporated into the analysis of the roll linkage. Figure 3-6 shows the parts included in the RSSR linkage connecting the roll servo to the right foot. The symbolic representation of the linkage is pictured in Figure 3-7.

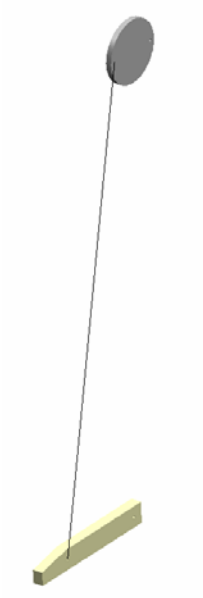


Figure 3-6. Parts included in the roll linkage on the right side of the robot.

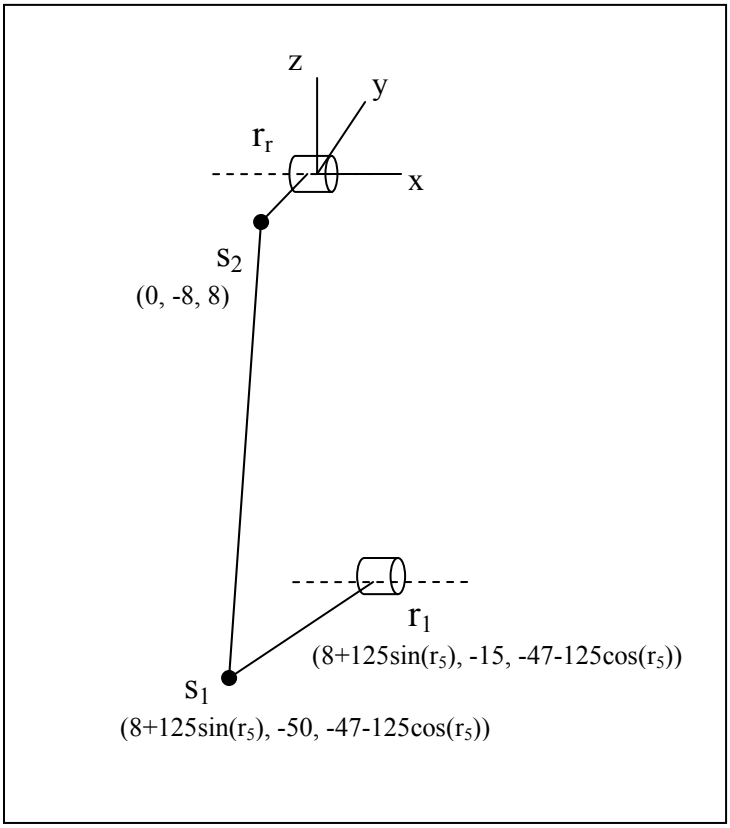


Figure 3-7. Symbolic representation of the RSSR linkage connecting the roll servo to the right foot. Initial dimensions are displayed for zero rotation of the r_r and r_1. However, the rotation of r_5 must be considered in the initial dimensions. For each position of r_5, a separate set of initial dimensions must be constructed.

A process similar to the one used to solve the pace linkage was followed to solve the RSSR linkage with roll servo. For each position of the hip angle, r_5, from the pace analysis, a separate set of initial dimensions was assigned to the joints. The location of the ankle joint, r_1, was determined by the angle of r_5. After the location of r_1 was set, a series of roll angles were introduced. The coordinates of the spheric joint, s_2, on the servo arm were calculated by

$$s_2 = \begin{bmatrix} 0 & -8mm\cos\theta_r & 8mm\sin\theta_r \end{bmatrix} \tag{3.3}$$

where θ_r is the angular position of the roll servo. The location of the spheric joint on the foot, s_1, was found according to

$$s_1 = \left[7 + 125mm\sin\theta_5 \quad -15 - 35mm\cos\theta_1 \quad -47 - 125mm\cos\theta_5 + 35\sin\theta_1\right] \qquad (3.4)$$

where θ_5 is the angle of the hip joint from the pace linkage and θ_1 is the angle of the ankle joint. The same brute force method was used to adjust θ_1 until the distance between the spheric joints matched the length of the connecting wire. Solving the angle of the ankle joint yielded the tilt angle of the robot when θ_1 was positive. A positive angle indicated that the robot was tilting to the right and was supported by the foot being analyzed. If the solution produced a negative angle, the robot was tilting to the left, and the angle from the RSSR solution for the left side would determine the tilt angle. The relationship between the position of both servos and the resulting tilt angle is depicted in as a three dimensional surface.

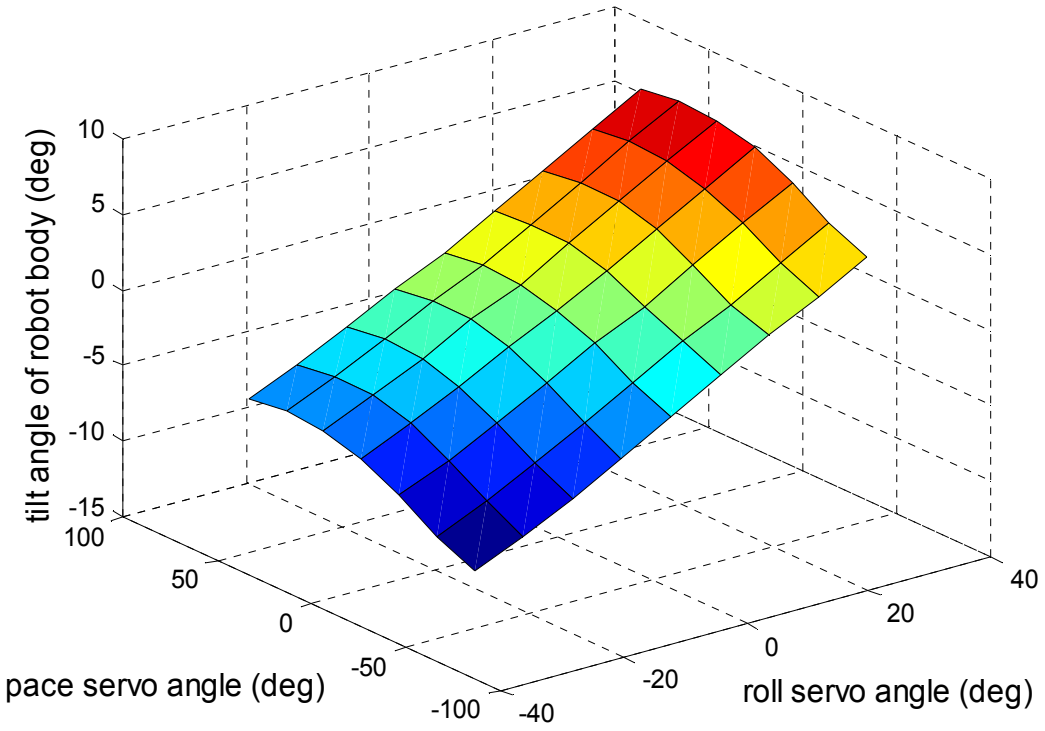


Figure 3-8. Three-dimensional surface showing the resulting tilt angle for a given pair of pace and roll servo positions.

Limitations of the joint motions were found empirically. The wire between the pace servo and leg reached a physical limit. For pace angles with a magnitude of approximately 60^o or greater, the wire pulled against the leg. The pulling against the wire loosened the joint between the wire and the leg. Restricting the pace angle magnitude to 55^o provided a safe range that avoided interference. The limit of the roll angle was set to provide the fastest straight-line walk. To walk in a straight line, the swing foot had to lift off of the ground. If the foot dragged, the robot could turn. On the other hand, raising the foot far above the ground would waste time. Whether the swing foot barely stayed above the ground or swung high in the air, the same straight walk would result. The only difference between the two motions would be the time required to raise and lower the foot. Therefore, the minimum roll angle that would lift the swing foot would allow the robot to walk fastest. A roll angle of $\pm 32^o$ was found to lift the swing foot just enough to avoid dragging and turning.

3.2 - Joint locations

After solving the RSSR linkages, the joint locations were solved based on the tilt and stride angle of the robot. The coordinates of each joint were defined relative to a coordinate frame attached to the robot body as shown previously in Figure 3-1 and Figure 3-2. Later, the robot coordinate frame was repositioned to yield the absolute coordinates of the joints. The origin of the robot coordinate frame was assigned to the ankle of the foot in contact with the ground. When the right foot was in contact with the ground, the origin was assigned to the joint r_1. When the left foot was in contact with the ground, the origin was assigned to the joint r_{10}. For cases when both feet were in contact with the ground, the origin was placed on the side to which the robot was leaning. If the robot stood fully upright, the origin was placed arbitrarily at

the left ankle. From the origin, rotation about the x-axis was considered tilt angle. Rotation of the legs about the hip joints was considered stride angle.

To begin, the positions of the joints were measured with the robot standing upright. The upright joint coordinates measured in millimeters were

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ -13 & -2 & 8.5 \\ -13 & -2 & 8.5 \\ -64 & -2 & 18 \\ -64 & -2 & 18 \\ -64 & 29 & 18 \\ -64 & 29 & 18 \\ -13 & 29 & 8.5 \\ -13 & 29 & 8.5 \\ 0 & 27 & 0 \end{bmatrix} \tag{3.5}$$

where the vector on the left side of the equation lists the revolute joints and the matrix to the right lists the x,y,z coordinates in the robot coordinate frame. Then changes in the joint positions were analyzed by first incorporating the stride angle and then including the tilt angle. Because the links connecting the hip and ankle joints formed a parallelogram, the angle of rotation about the hip axes mirrored the rotation about the ankle axes. Thus, the stride angle was equated to the rotation about the ankle axis. Then the positions of the hip joints were solved. For instance, the position of the hip joint, r_3, could be determined after positioning the stride angle. Assuming the origin was at the right foot, the hip joint, which had an upright location of

$$r_3 = \left(-13, \quad -2, \quad 8.5\right) \tag{3.6}$$

could be considered to move in an arc around the ankle axis, r_2. Because the stride angle, θ_s, was reflected in the rotation about the ankle axis, the position of the hip angle could be calculated according to

$$r_3 = \left(-13 + 125\sin\theta_s, \quad -2, \quad 8.5 + 125\cos\theta_s\right) \tag{3.7}$$

.

On the opposite side from the origin, the hip joints were offset by the width of the torso. The hip joint, r_8, on the left side from r_3 had a position defined by

$$r_8 = \left(-13 + 125\sin\theta_s, \quad 29, \quad 8.5 + 125\cos\theta_s\right) \tag{3.8}$$

where the torso width of 31 mm has shifted the y-coordinate. Then the positions of the ankles on the left side were calculated by doubling the distance of the hip joints from the origin. The distance was doubled because the connection of the pace servo forced the hip joints to turn in opposite directions on either side of the robot. If the hip joints moved a distance in the x-direction, the ankles on the opposite side moved the same distance from the hip joints. Therefore, the distance that the opposite ankles traveled in the x-direction was twice the distance of the hip joints.

After all the joint positions were solved with respect to changes in the stride angle, the tilt angle was considered. Changes in the tilt angle caused a rotation of the robot about the ankle joint aligned with the x-axis. A rotation matrix was computed to rotate all the coordinates about the x-axis by the tilt angle. The rotation matrix R_x was given by

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_t) & -\sin(\theta_t) \\ 0 & \sin(\theta_t) & \cos(\theta_t) \end{bmatrix} \tag{3.9}$$

where θ_t is the tilt angle. To apply the rotation, R_x was multiplied by the joint coordinates. Two sets of coordinates were solved for each possible reference frame. With the origin of the robot coordinate frame assigned to the right foot, the set of joint locations was

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{bmatrix} = R_x \begin{bmatrix} 0 & 0 & 0 \\ -13 & -2 & 8.5 \\ -13+125\sin\theta_s & -2 & 8.5+125\cos\theta_s \\ -64 & -2 & 18 \\ -64+125\sin\theta_s & -2 & 18+125\cos\theta_s \\ -64+125\sin\theta_s & 29 & 18+125\cos\theta_s \\ -64+250\sin\theta_s & 29 & 18 \\ -13+125\sin\theta_s & 29 & 8.5+125\cos\theta_s \\ -13+250\sin\theta_s & 29 & 8.5 \\ 250\sin\theta_s & 27 & 0 \end{bmatrix} \tag{3.10}$$

where again the vector lists the joint names and the matrix lists the joint coordinates. When the origin was assigned to the left foot, the joint locations were given by

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{bmatrix} = R_x \begin{bmatrix} -250\sin\theta_s & -27 & 0 \\ -13-250\sin\theta_s & -29 & 8.5 \\ -13-125\sin\theta_s & -29 & 8.5+125\cos\theta_s \\ -64-250\sin\theta_s & -29 & 18 \\ -64-125\sin\theta_s & -29 & 18+125\cos\theta_s \\ -64-125\sin\theta_s & 2 & 18+125\cos\theta_s \\ -64 & 2 & 18 \\ -13-125\sin\theta_s & 2 & 8.5+125\cos\theta_s \\ -13 & 2 & 8.5 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

where the upright coordinates have shifted by 27 mm in the negative y-direction to account for the width between the left and right ankles, and the effect of the stride angle on the x-coordinates has reversed magnitude.

Finally, the joint positions in the robot coordinate frame were related to a global coordinate frame. Rotating and translating the robot coordinate frame moved the coordinates to the robot's absolute location. A rotation matrix was used to turn the coordinates to match the heading, θ_h, of the robot. The rotation matrix expressed as

$$R_z = \begin{bmatrix} \cos(\theta_h) & -\sin(\theta_h) & 0 \\ \sin(\theta_h) & \cos(\theta_h) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

caused a rotation about the vertical axis. In the direction of the robot heading, the origin of the robot coordinates was translated by the distance traveled during each step. The translation of the origin was calculated according to

$$O_f = \begin{bmatrix} O_{ix} + s\cos\theta_h & O_{iy} + s\sin\theta_h & 0 \end{bmatrix} \tag{3.13}$$

where O_f was the translated position of the origin, O_i was the position of the origin before being translated, and s is the distance traveled during the step. The joint coordinates were then defined relative to the translated origin to yield the absolute coordinates of the robot's joints.

In review, the global coordinates of each joint were determined through kinematic analysis. From the expected location of the stance foot, a robot coordinate frame was established. Tilt and stride angles for the robot were derived from RSSR analysis. Then every joint location was solved based on the tilt and stride angles. Typically, kinematic equations

would be used to develop dynamic equations. Because the robot studied in this research maintained static stability, the kinematics were not used in the control system. However, the kinematics were used in testing the control performance. The expected locations of the feet were calculated via the kinematics. Then the actual footsteps of the robot were compared to the expected values to determine the accuracy of the robot. In future research, these kinematic equations could be used in performing dynamic analysis on the robot.

Chapter 4 - Robot movement

Feedforward control required predictions of how the robot would move. To learn how the robot moved, various walking motions were studied. Experiments were conducted to observe the robot walking with the roll servo in different positions. These walking tests revealed how the roll servo affected the distance and direction in which the robot moved. A relationship was developed between the servo position and the resulting step length and turning angle of the robot. Repeating the movements exhibited significant error from one iteration to the next. Therefore, feedforward control would be insufficient for accurate trajectory tracking. However, the predictions of the movements could be used in concert with a feedback mechanism to produce accurate tracking.

4.1 - Overview of how the robot moved

Having two degrees of freedom, the robot could move by tilting to the side and swinging its legs forward and backward. In this research, the term referred to as tilting indicates a motion similar to a human leaning to the right or left. Tilting created a rotation about the ankle axis as shown in Figure 4-1. The term striding describes the legs swinging about the hip axes. Figure 4-2 depicts the striding action. All robot motions consisted of tilting and taking strides at various speeds. The position of the roll and pace servo motors determined the tilt and stride angles. Given the roll and pace servo positions, the location of every joint could be found via the kinematic analysis. Thus, the two servos were able to set the body orientation completely. Also,

the servo speed determined the speed with which the robot moved. Coordinating motions of the servos generated the walking motions.

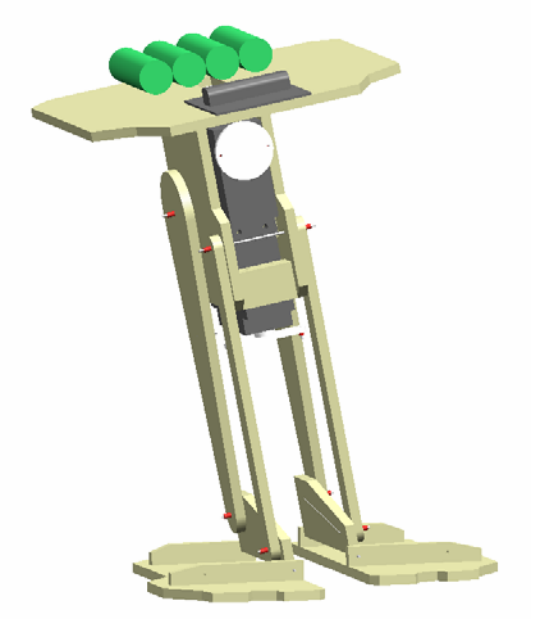


Figure 4-1. Tilting actuated by the roll servo. The robot is shown here tilting onto its right leg while raising the left leg.

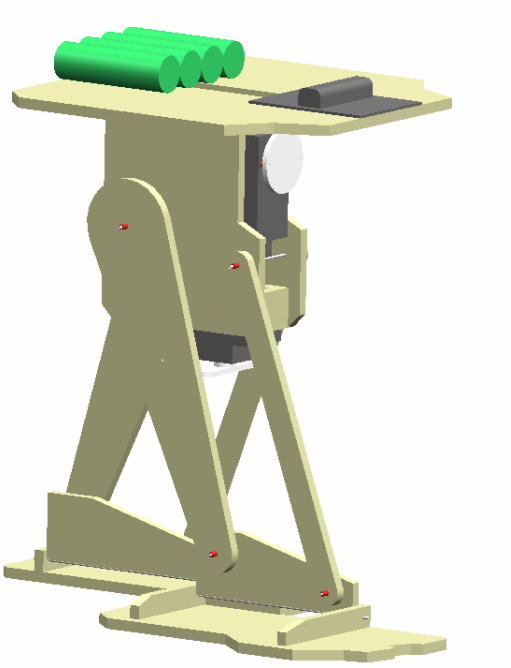


Figure 4-2. Striding actuated by the pace servo. The robot is shown here with the right foot extended forward in the lead position with the left foot extended backward in the trailing position.

Studying the robot locomotion began with controlling the robot to walk in a straight line. The Stamp I microcontroller arrived preprogrammed to cause the robot to walk forward. To produce a forward walk, the legs alternately lifted and swung forward. Turning the roll servo caused the robot to tilt toward one side while lifting the leg on the opposite side. Then the pace servo turned to swing the raised leg. Turning the roll servo back in the opposite direction tilted the robot onto the foot that had just swung. Repeating these movements created a waddling walk reminiscent of a penguin. Backward walking was achieved by reversing the direction of the pace servo.

Turning was achieved by dragging the feet along the ground. Rather than tilting to raise one foot above the ground, as in straight-line walking, both feet remained in contact with the ground during turns. For example, the right leg during forward walking would lift, swing forward, and step down again. To turn right, the right leg would not lift completely above the ground. Instead of swinging forward, the right leg dragged forward. Because of the way that the robot was constructed, the pace servo would pull backward on the left leg while pushing forward on the right. The frictional forces acting in opposite directions on the feet would force the robot to rotate clockwise.

For this research, restrictions were placed on the set of possible motions the robot could perform. Adjustments to the servo positions occurred independently for the roll and pace servos. Performance could have been improved by moving both servos simultaneously. Turning the pace and roll servos at the same time could increase the walking speed. Before the roll servo finished tilting the robot, the pace servo could begin the leg swing. This would reduce the time each servo would spend waiting for the other servo to stop moving. Each step would require less time and the walking speed would increase. However, the swing foot would drag if it started

moving before the tilting motion was complete. Dragging the foot would cause the robot to turn. As the tilt motion continued, the friction under the feet would change. As a result, the varying friction would affect the turning rate. Analysis was not performed to predict the way in which turning would be affected and remains outside the scope of this research.

Another restriction placed on the robot was the exclusive use of full-length strides. Variable stride lengths could have been incorporated. Shorter strides could have been used to produce shorter steps or smaller turning angles. Controlling the stride length would be important for collision avoidance. A shorter step might be desired to avoid hitting an obstacle. In this research, however, the walking surface contains no boundaries or obstacles. Striding always consisted of switching the legs from full extension with one leg forward and the other backward to full extension with the opposite legs forward and backward. Each step included a motion of the roll servo, and then a motion of the pace servo to switch the leading and trailing legs.

4.2 - Factors affecting robot movements

The turning direction was affected by the direction that the pace servo turned to produce a stride. It was found that, striding to switch the position of the legs determined the direction in which the robot turned. The turning direction was related to the orientation of the legs before the turn. Because full strides were used for all steps, each turn would begin with one leg fully extended in front of the robot and one leg behind. Extended forward, the lead leg dragged backward during the turn. At the same time, the trailing leg slid forward. If the right leg began in the lead position, the right leg moved backward and the left leg came forward. The frictional forces working in opposition to the sliding feet created a counterclockwise moment. As a result,

the robot turned counterclockwise. If the left leg began in the lead position, the forces reversed and the robot turned clockwise.

Adjusting the pace servo affected the magnitude of the robot's turning angle. By setting the tilt angle of the robot, the position of the roll servo could vary the weight distribution between the feet. By pulling up more on the outstep of one foot and pushing down more on the outstep of the other foot, additional weight would shift onto the foot whose outstep was lifted. With greater weight applied to a foot, the normal force acting between that foot and the ground would increase. Thus the frictional force, directly related to the normal force, would increase under that foot. Changing the relative frictional force between the two feet altered the turning angle. The largest turning angles tended to occur when the weight was more evenly distributed and the friction more symmetrically opposed.

Also, the roll position influenced the distance traveled during a turning step. With the tilt angle making frictional forces between the feet equal and opposite, the torso of the robot could be expected to rotate in place. While the feet would slide an equal distance in either direction, the torso would remain in the same location. Perfect rotation without displacing the torso was never achieved in this research. However, the expected relationship between the tilt angle and torso displacement during turns was observed. When the roll servo positioned the robot nearly upright, the torso displacement decreased. Tilting the robot more to one side resulted in larger displacements of the torso. If the robot tilted to apply more weight onto one foot, that foot tended to slide less, and the other foot tended to slide more. When most of the weight was supported by the leading leg before the turn, the robot stepped forward while turning. If the trailing leg held most of the weight, then the robot stepped backward.

4.3 - Tests to measure possible movements

Tests were performed to measure the turning angle produced by strides with constant roll positions. The foot initially in the lead position was related to the turning direction. Therefore, testing of the turning angle produced two sets of data. One set was collected for turns with the right foot initially in the lead position. The seconds set corresponded to turns beginning with a left-foot lead. If the right foot extended forward before the turn, the robot turned counterclockwise. Starting a turn with the left foot in front generated a clockwise rotation. For each configuration of the lead foot, a series of turns were produced for different positions of the roll servo.

While testing for the turning angle, the displacement of the robot also was measured. Displacement across the ground was recorded in terms of the step length. Because the feet slid along the ground, the step length was not always equal to the stride length. Every turning motion involved a complete stride. However, the sliding feet could leave the torso in the same position. Thus, the full stride could result in zero step length. In this research, the step length was defined as the distance from the position of the lead foot before the turn to the position of the other foot after the turn. This distance was measured in the direction of the robot's heading after the turn.

When a step was taken in a straight line, the step length matched the stride length. Figure 4-3 illustrates the way in which step length was defined. The feet with the black outline depict the robot initially standing with the right foot forward. The step length, s, is shown after the robot has stepped forward with the left foot. Every stride moves one leg sixty millimeters in front of the other. In this case, the step length is shown equal to the stride length.

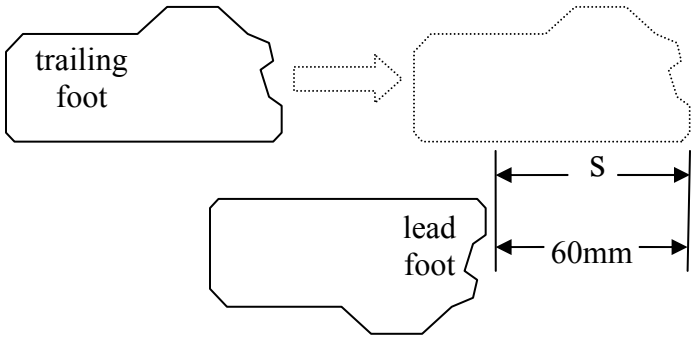


Figure 4-3. Forward stepping where the step length and stride length are equal. The right foot is shown in the lead position. The left foot begins in the trailing position and moves forward into the lead position. Measured from the position of the initial lead foot to the final lead foot, the step length is indicated by the distance s.

In contrast to stepping straight forward, a turning step would produce a step length that differed from the stride length. Figure 4-4 shows how a left turn could affect the step length. As in Figure 4-3, the feet are shown initially with a right foot lead. The faint outlines show the final orientation of the feet. In their final positions, the feet have rotated counterclockwise. To produce a turn, the feet must have dragged on the ground. While the left foot dragged forward, the right foot slid backward. Figure 4-5 shows that the stride length remained the full sixty millimeters as in stepping straight ahead. Because the right foot slid backward, the resulting step length was less than the stride length.

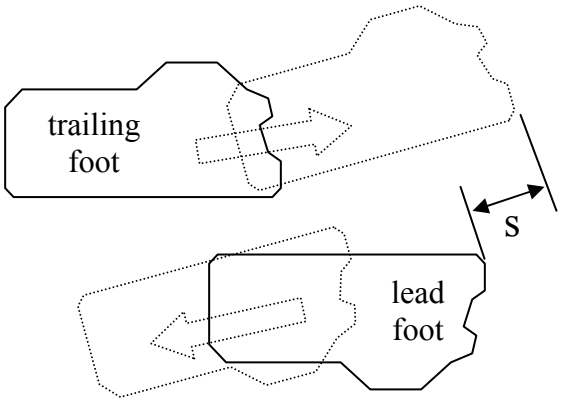


Figure 4-4. Counterclockwise turning that results in a step length less than the stride length. The faintly outlined feet and the arrows indicate that the left foot has moved forward and the right foot backward. The step length is shown as the distance from the initial lead foot to the final position of opposite foot. Also, the distance is measured in the direction that the feet are pointing after the turn.

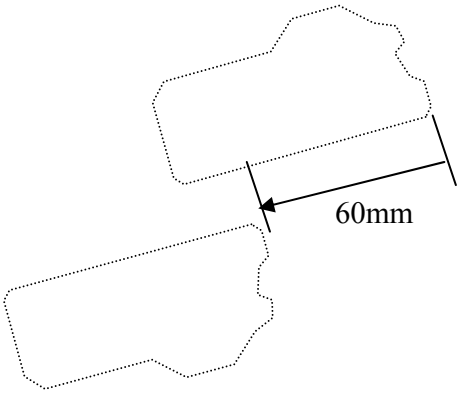


Figure 4-5. The same scenario shown for counterclockwise turning except that the final positions of the feet are displayed solely. This image highlights that the stride length separating the feet remains 60 mm.

For each turning step, marks were placed on the walking surface to record the movement of the lead foot. The initial location of the lead foot was marked by drawing a pencil line around one corner of the foot. Another pencil line drawn along the side of the foot marked the orientation. In Figure 4-6, the markings are illustrated for the counterclockwise turn considered in Figure 4-4. The markings were made at the right foot, which began in the lead position. After turning, the same marks were repeated for the same foot at its final location and orientation. Figure 4-7 shows the markings added at the right foot after the turn was complete.

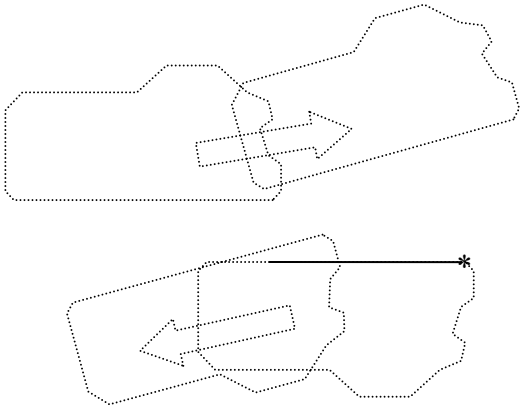


Figure 4-6. Counterclockwise turning with the markings made to record the initial location and orientation of the lead foot. The asterisk symbolizes the location marker at the corner of the foot. The black line depicts the line drawn along the side of the foot.

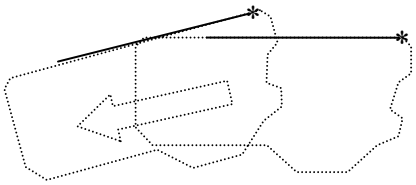


Figure 4-7. Counterclockwise turning with markings at the right foot before and after the turn.

Measurements of the turning angle and step length were taken by hand. After extending the linear markings until they intersected, the angle could be measured. A protractor was used to measure the turning angle with a resolution of one degree. Counterclockwise turning was assumed to occur in the positive direction. Step length was measured using the initial and final location markers. These markers directly indicated the location of the foot that began in the lead position. However, the step length included the end position of opposite foot. This end position was inferred. A new mark was made extending 60 mm in front of the final position marker. This distance corresponded to a full-length stride. Then the step length was measured from the initial location marker to the new, inferred marker.

The case of the counterclockwise turn can illustrate the measurement process. The linear markings were extended as shown by the dashed lines in Figure 4-8. Overlaying a protractor on the extended lines allowed the turn angle, θ, to be measured. A new asterisk is shown 60 mm in front of the final position marker on the right foot. During the stride, the left foot would have extended a full stride length in front of the right foot. Therefore, the new marker was placed 60 mm ahead of the right foot to indicate the final position of the left foot. Finally, the step length, s, was measured as the distance from the initial location of the right foot to the final location of the left foot.

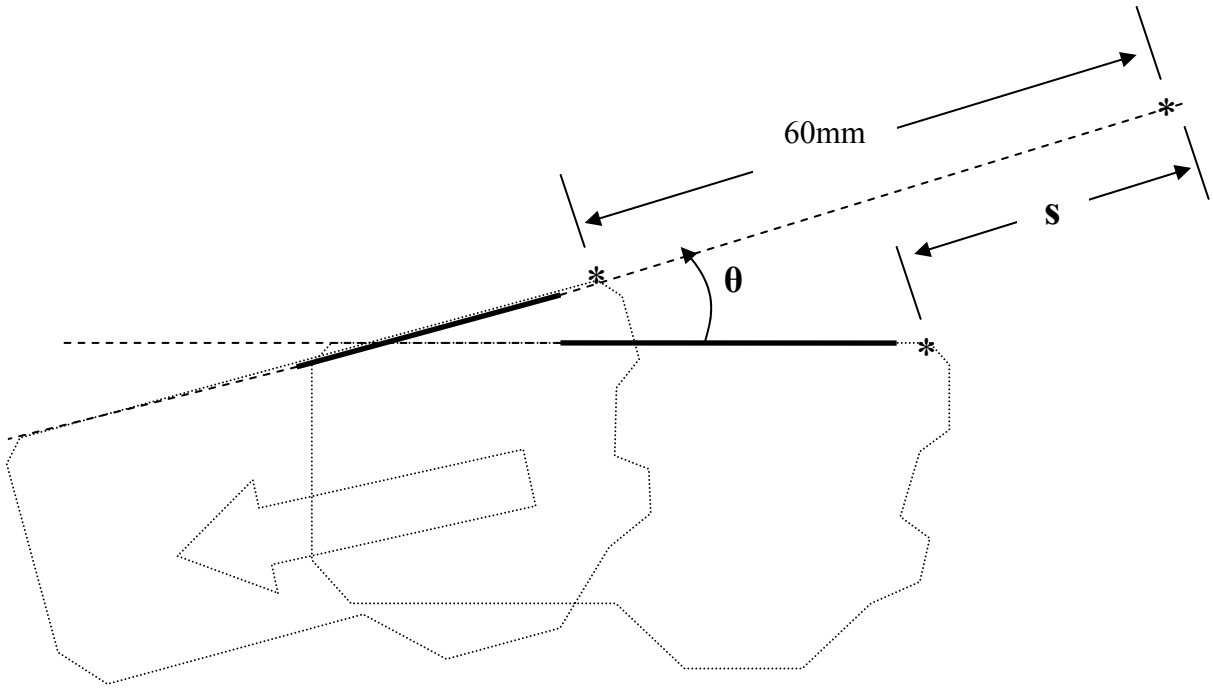


Figure 4-8. Measurement of the turning angle and step length during the counterclockwise turn. The turn angle is indicated by θ. The arrow on the corresponding arc shows the direction in which the angles were measured. Because the turn was counterclockwise, the turning angle was positive. The 60 mm dimension shows the inferred extension of the left foot after the turn. From the marker on the right foot to the inferred marker on the left foot, the step length, s, was measured.

4.4 - Results of the movement testing

The turning angle was measured for turns with various positions of the roll servo. Before each turn, the roll servo was moved to the desired position. Then during the turn, the position of the roll servo remained constant. Two separate sets of data were collected, one set for turns beginning with the right foot in the lead position and the other set for turns starting with a left foot lead. Positive turning angles occurred when the turn began with a right foot lead. The positive angles that were measured are plotted in Figure 4-9. A roll position of zero degrees corresponded to the robot standing upright. When the roll servo tilted the robot farther from the upright position, smaller turning angles resulted. Because the roll servo had tilted the robot, the

lifted leg was freer to swing. Less friction acted under the swing foot. Thus, a weaker turning moment was produced to turn the robot. Turning angles increased as the roll servo position moved closer to zero and the robot stood more upright. Larger turning angles were expected near the upright position, because the robot's weight was more evenly distributed between the feet. With both feet sharing the weight and dragging the ground, the opposing frictional forces generated a stronger turning moment. Therefore, the turning angles increased. Starting with a left foot lead produced negative turn angles. Measurements recorded for turns starting with a left foot lead are plotted in Figure 4-10. Again the magnitude of the turning angles increased as the roll servo tilted the robot less.

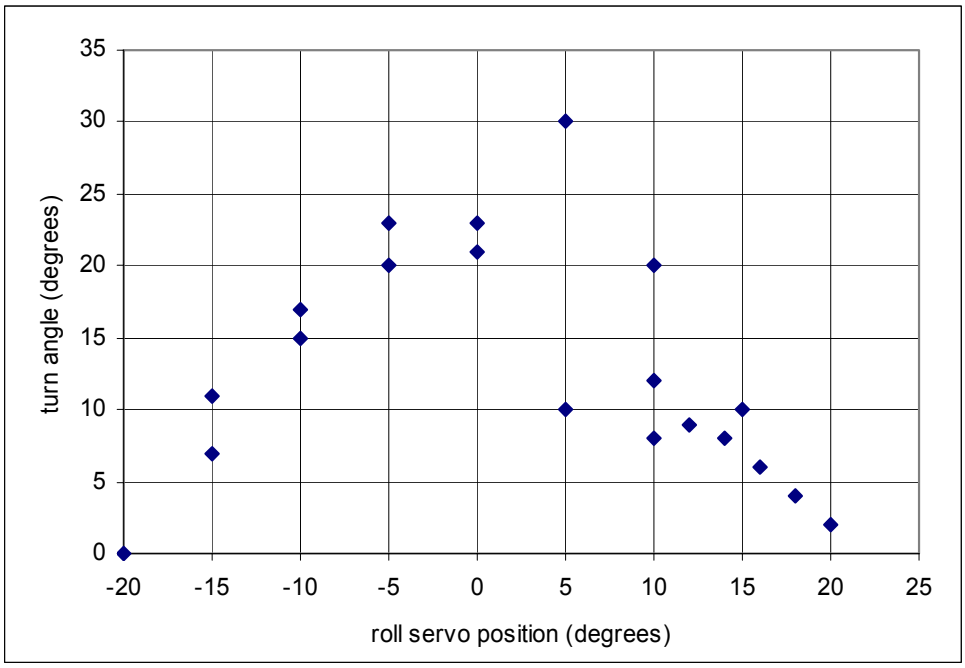


Figure 4-9. Turn angle produced when the robot was oriented initially with the right foot forward. Pulling the right foot backward while pushing the left foot forward created counterclockwise turns (positive turn angles).

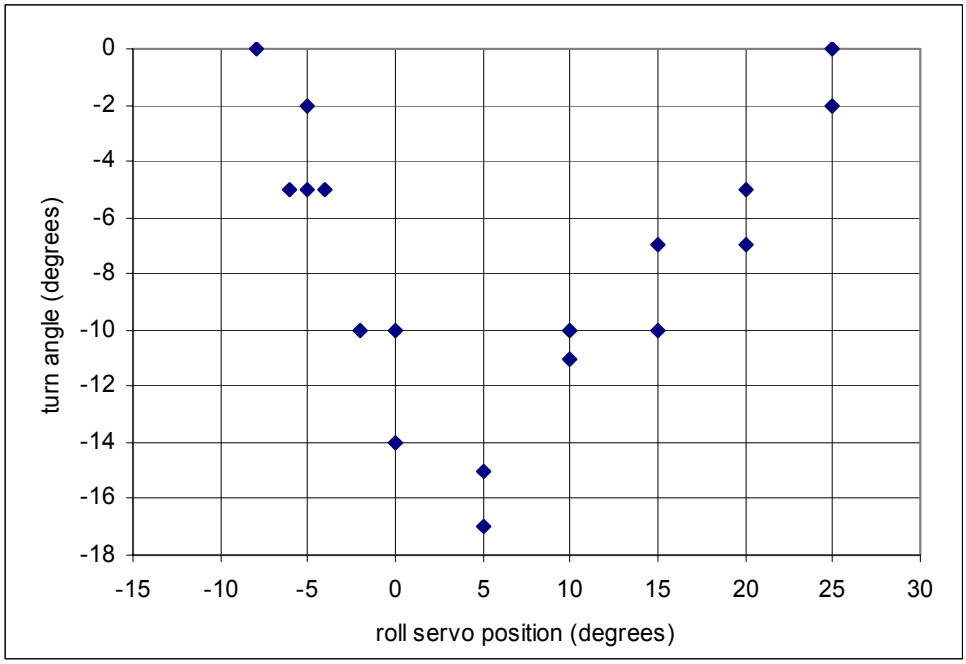


Figure 4-10. Turn angle produced when the robot was oriented initially with the left foot forward. Pulling the left foot backward while pushing the right foot forward created clockwise turns (negative turn angles).

The plots of the turning angle reveal imprecision in the robot's execution of the turns. In the case of turning from a right foot lead, the data points scatter at the 5 and 10° positions of the roll servo. In particular, the two turns performed with the roll servo at 5° produced angles measuring 30° and 10°. Assuming that the average angle of 20° should have been expected, the actual angles varied from this expected value with 50% error. The resolution of the protractor used in the testing measure the angles with a resolution of 1°. Therefore, the high error was considered to be an unpredictable action of the robot. Such a high error value would make feedforward control unreliable. Feedback control could be used to correct for errors in the turning angle. Because the use of sensory feedback remains out of the scope of this research, creating a feedback control system for the robot is recommended for future research.

During the same tests for the turning angle, measurements were recorded for the step length. Plots of the step length at various turning angles are presented in Figure 4-11 and Figure

4-12. Figure 4-11 shows the step lengths for turns beginning with a right foot lead. In Figure 4-12, the data for turns starting with the left foot forward are shown. When the roll servo tilted the robot more onto the lead leg, the step lengths occurred in the forward direction. When the robot was tilted onto the trailing leg, the robot stepped backward. For each position of the roll servo, the corresponding step lengths appear to match closely. However, a high degree of imprecision remains in the step lengths. The highest degree of error occurred during turning from a left foot lead with a roll position of 10^{o}. At this position, the step lengths produced were 5 and 10 mm in the backward direction. The error about the average value of -7.5 mm was 33%. Again, feedback control is recommended to solve the robot's error problem in future research.

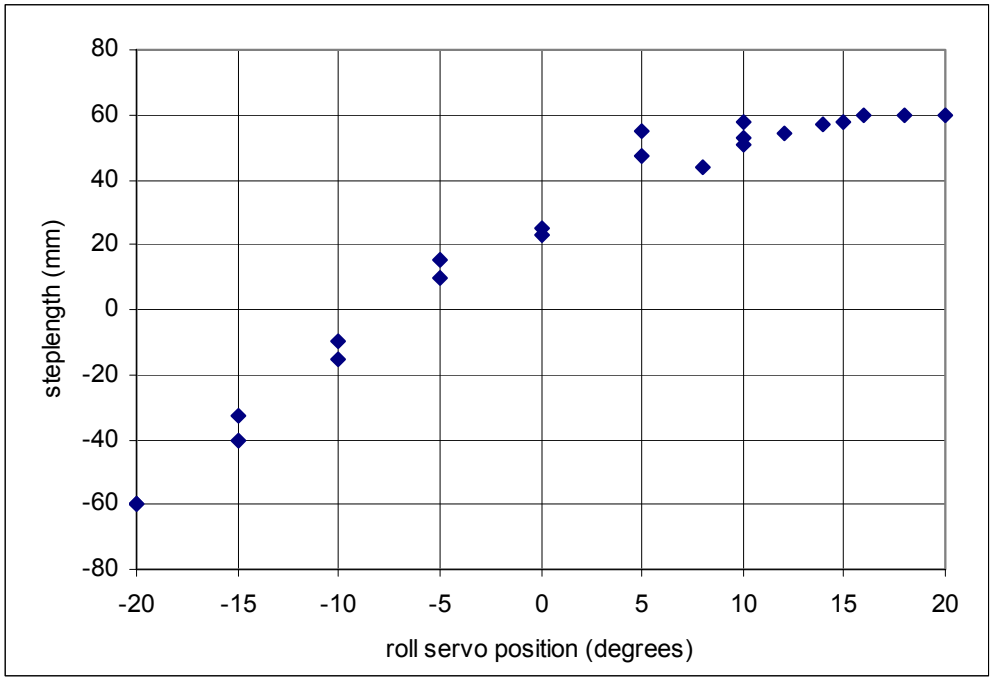


Figure 4-11. Step length measurements during turns beginning with a right foot lead.

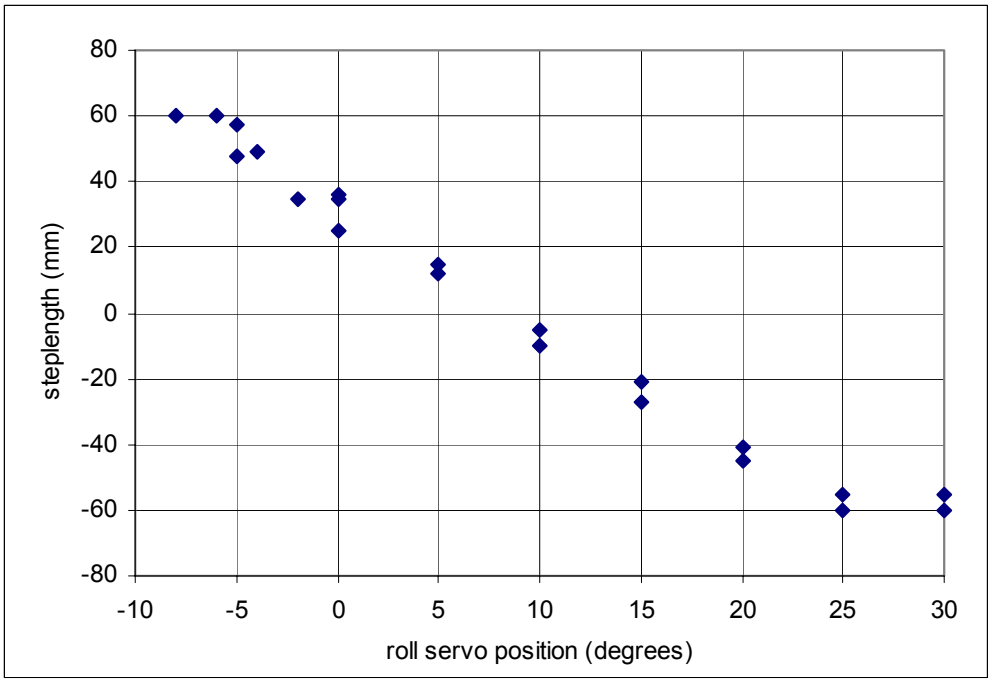


Figure 4-12. Step length measurements during turns beginning with a right foot lead.

4.5 - Set of possible movements

After the turning angles and step lengths were recorded, the average values at each position of the roll servo were tabulated. The resulting average values of the step length and turn angle are listed in Table 4-1. Data for the turns beginning with the right foot forward is given in the center section of the table. On the right side, the data for the turns starting with a left foot lead is listed. The values listed in this table were used later to control the motion of the robot. Knowing the relationship between the roll position and the resulting robot motion was required before the control system could be designed. Based on this relationship, the roll servo could be adjusted to produce the desired robot motion. This control algorithm will be discussed in the chapter on the control system.

Table 4-1. Step length and turning angle during strides with the roll servo in various positions.

roll servo angle (degrees)	initial right foot lead		initial left foot lead	
	step length (mm)	turn angle (degrees)	step length (mm)	turn angle (degrees)
-15	-37	9	60	0
-10	-13	16	60	0
-5	13	22	54	-5
0	24	22	32	-12
5	51	20	14	-16
10	54	13	-8	-11
15	58	7	-24	-9
20	60	0	-43	-6

Counterclockwise turning produced angles of greater magnitude than clockwise turning produced. Larger turning angles indicate that the turning moment was stronger. Differences in the bottom surfaces of the feet may have caused the difference in turning moment. Possibly the grain of the balsa wood made the feet slide more easily in one direction than the other. The feet may have generated the more friction when the right foot slid backward and the left foot dragged forward. If this were the case, the turning moment would be stronger in the counterclockwise direction than in the clockwise direction.

To ensure that the robot could turn in the desired direction, the orientation of the feet needed to be correct. Because all strides moved both legs to their farthest front or rear extension, each turning motion began with the legs fully extended. The legs could only move from their initial extension to the opposite extension. If the left foot were the lead foot, then the stride would move the left foot backward and the right foot forward. With the left foot as the lead foot, the robot could turn clockwise only. With a right lead, the robot could turn counterclockwise only. In situations where the previous stride did not leave the robot oriented in such a way that it could turn in the desired direction, a step was required to switch the leg orientation before executing the turn.

For cases when the orientation needed to be switched, the step length data was altered to include stepping before the turn. In the data, the stepping and turning were considered to be part of the same turn. If a forward step length was desired during the turn, the robot stepped forward to switch the lead leg. If the robot needed to move backward during the turn, the orienting step was taken backward. Table 4-2 lists the step lengths and turning angles adapted to include an orienting step. The turning angle did not change, but the displacement was decreased by 60 mm when a backward stride was used to switch the lead foot and increased by 60 mm when a forward stride was used. Negative step lengths occurred when a backward stride preceded the turn. When a forward stride preceded the turn, a positive step length resulted.

Table 4-2. List of the step length and the angle turned when the lead foot was switched before the turn was executed.

		left foot lead				right foot lead		
roll servo angle (degrees)		step length (mm)	step length (mm)	turn angle (degrees)		step length (mm)	step length (mm)	turn angle (degrees)
-15		-97	23	9		0	120	0
-10		-73	47	16		0	120	0
-5		-47	73	22		-6	114	-5
0		-36	84	22		-28	92	-12
5		-9	111	20		-46	74	-16
10		-6	114	13		-68	52	-11
15		-2	118	7		-84	36	-9
20		0	120	0		-103	17	-6

By including the effect of the orienting step, a complete set of possible robot movements was created. If the robot began with a left foot lead, clockwise turns could be performed by a single stride. Counterclockwise turns required a preparatory step to bring the right foot into the lead position. With the right foot in the lead position, a single stride could turn the robot counterclockwise. Clockwise turns required an orienting step to move the left foot forward

before turning. Either turning direction could be coupled with a forward or backward step length.

Studying the robot movements yielded predictions of step length and turning angle. These predictions were used in a feedforward control system. A set of possible movements was created for various positions of the servos. Selecting from this set of movements, the control system attempted to make the robot follow a desired path. The random error measured in the stepping tests made feedforward control unacceptable for autonomous applications. With each step, the error compounded with the error from the previous step. This compounding effect caused the robot to stray increasingly far from the desired path. However, future research could apply the list of robot movements in a feedback controller.

Chapter 5 - Linear velocity and angular velocity variation

After finding ways to control the robot displacement and direction, additional study led to a method for controlling the robot velocities. The goal of the control was to track a trajectory that was defined in terms of linear and angular velocities. Therefore, control over the robot's velocities was desired. In the previous chapter, the step lengths and turning angles produced by the robot were discussed. Setting the time to reach the step length and turning angle would determine the linear and angular velocity of the robot. By controlling the speed of the servos, the desired robot velocities could be produced. Thus, a relationship was developed between servo commands and the desired linear and angular velocities.

In this research, it was assumed that a higher level controller has translated a trajectory into linear and angular velocities. The type of nonholonomic system represented by the robot has been proven to be controllable [22]. In a parallel example of controlling a unicycle, the motion could be defined in terms of the rolling speed and turning speed. A lower level control would be the rider pedaling at a certain rate to achieve the desired rolling speed. Also, the rider would need to twist his or her body and lean to produce the desired turning speed. Analogous to the unicycle problem, a method was sought to make the biped robot move with desired linear and angular velocities.

5.1 - Commanding the servo motors

The roll and pace servo motors operated on servo control pulses. From the Stamp I microcontroller, pulses were sent to the servo motors. The duration of each pulse determined the goal position of the servo. After receiving a pulse, the servo would move a short distance

towards the goal. An example of servo control pulses is shown in Figure 5-1. Two pulses are plotted with respect to the elapsed time. The first pulse duration measured 1.51 ms. For the pace servo, the center position corresponded to a 1.51 ms pulse. When the microcontroller sent this pulse length to the pace servo, the servo would move toward its center position. This center position was determined as the point at which the servo held the legs at zero stride angle. As shown in the diagram, a second pulse was generated after a 10 ms pause. Separating consecutive pulses, the pause time allowed the servo to move toward the previous goal and prepare to receive the next pulse. In the diagram, the duration of the second pulse measured 1.52 ms. Lasting longer than the first pulse, the second pulse corresponded to a different goal position. If this new pulse were sent to the pace servo, the servo would move toward a position slightly clockwise from center.

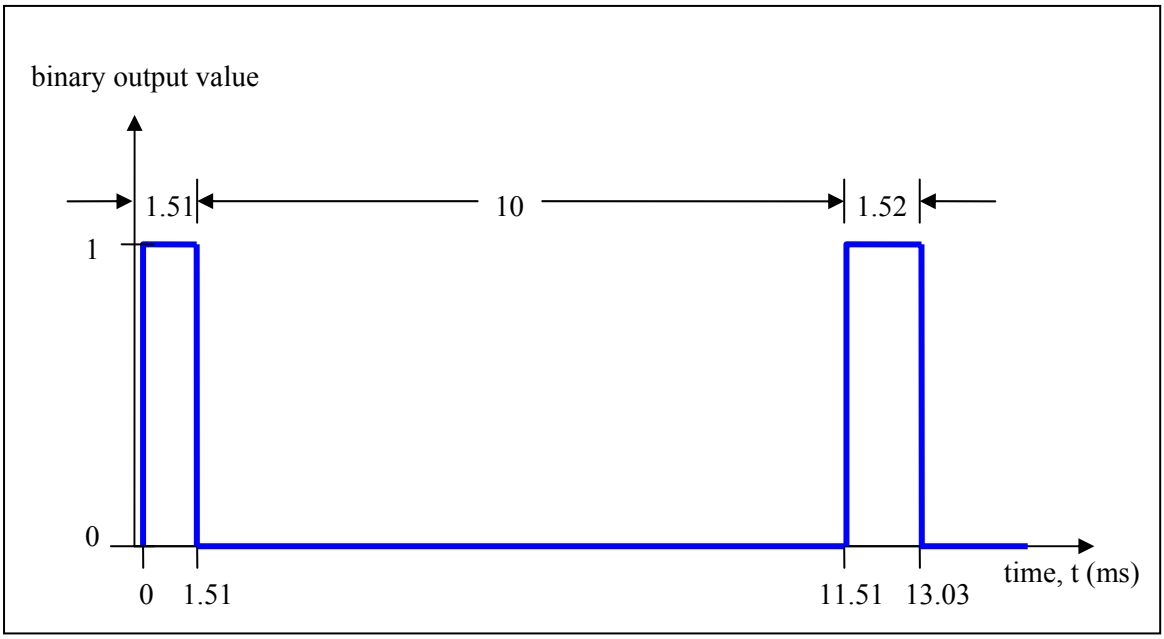


Figure 5-1. Servo control pulses illustrating the pulse duration and pause time between pulses.

Although the servo position could be commanded, it was required that the servo motors reach the goal position for each pulse. Because feedforward control was used in this research, the control system needed to predict the position of the servos. However, if the servos were commanded to move too fast, they would lag behind. Then the servo position would become unknown. Each pulse was required to give a goal near the current position of the servo. If the goal was close enough, then the servo would move the full distance to the new goal position after receiving the pulse. If the goal were farther from the current position of the servo, then the servo would turn faster. However, if a new goal position was given that varied greatly from the current position, then the servo would turn only part of the distance to the new goal.

A test was conducted to determine the maximum allowed servo speed without having the servo lag behind the control pulses. Each servo was commanded to oscillate between two goal positions. The commands sent to the servo were given as a series of servo pulses. This test was repeated for various incremental changes between pulses. To begin, the pulses were varied by 10 μs from one pulse to the next. During each trial, the pulse increment increased by an additional 10 μs. As the step size between pulses increased, the servo turned faster. If the servo continued to reach the goal positions while oscillating, then the servo tracked the commands. When the servo fell short of the goals, the test indicated that the servo lagged behind the commands. A noticeable lag appeared when an increment of 60 μs was used. The ability of the servos to track an incremental change of 50 μs remained unclear. During testing of each increment of 40 μs or less, the servo tracked the commands. The 50 μs increment was not investigated further, primarily because the resulting speed compromised the static stability. When the roll servo was given commands using the 50 μs increment, the robot began to bounce

until it toppled over. Therefore, the maximum speed of the servos was limited to using the 40 μs increment between pulses.

With the Stamp I microcontroller, the Basic operation, PULSOUT, was used to send servo pulses. This operation produced a pulse length related to the number specified in the command. The resulting pulse length duration, τ, was determined according to

$$\tau = X * 10\mu s \tag{5.1}$$

where *X* represents the number included with the PULSOUT command. Each integer change in the command produced a 10 μs change in the servo pulse. Thus, the resolution of the PULSOUT command was 10 μs. As a result, the least possible change between consecutive servo pulses was 10 μs. The largest change between consecutive pulses considered in this research was 40 μs.

By adjusting the pause time between servo pulses, additional resolution in the possible servo speeds was achieved. The method of changing the speed by setting the increment between pulses has been discussed. In this method, only four speeds were available. Servo commands could be sent with 10, 20, 30, or 40 μs increments between pulses. However, greater resolution was desired in the servo speed. Testing on the affects of the increment between pulses was performed with a constant 10 ms pause between pulses. A pause of at least 10 ms was required to ensure that the servos would be ready for the next pulse. However, longer pauses could be used. Moving the robot faster involved controlling the servos to reach their desired positions in less time. Therefore, the shortest pause resulted in the fastest speed. On the other hand, slower motions were achieved by lengthening the pause duration. Because longer pauses caused the

servo to wait inactively for the next pulse, the average speed over the entire motion decreased. Adjusting the pause duration created an infinite set of possible speeds. Larger increments and shorter pauses resulted in faster servo speeds, while smaller increments and longer pauses resulted in slower speeds.

Although the pauses could be used to generate infinite possible speeds, a reduced set was created to achieve the smoothest motion. Each movement of the servo included a period of motion and a period of waiting for the next pulse. Long pauses between the pulses caused the servos to move with a jerking start and stop motion. For each desired speed, the shortest pause was used so that the smoothest motion could be achieved. To generate the fastest speed, the maximum incremental pulse change of 40 μs and the minimum pause of 10 ms was used. To slow the servo from this maximum speed, the pause was increased until the speed corresponded to an increment of 30 μs with a shorter pause. The shorter increment slowed the servo, but the shorter pause increased the speed. A list of possible increment and pause combinations was made to provide the shortest pause for each desired speed. This list is given in Table 5-1. For each setting of the increment, the table provides the maximum and minimum pauses used. The speeds corresponding to the combination of the increment and pause are listed in the rightmost columns. The speeds were calculated by

$$\dot{p} = \frac{i}{(i + \delta)} \tag{5.2}$$

where $\dot{p}$ is given in terms of the change in pulse command per second, i is the incremental change between servo pulses, and δ represents the pause time. When selecting a pause between the maximum and minimum values, the corresponding speed could be calculated.

Table 5-1. Possible settings for the servo speed. Minimum and maximum pauses are listed for each incremental change between pulses. The corresponding maximum and minimum speeds are given. The control system achieved a wide range of speeds by using pause times between the minimum and maximum values listed.

increment between pulse commands (μs)	maximum pause (ms)	minimum pause (ms)	minimum speed (change in pulse command per second)	maximum speed (change in pulse command per second)
10	255	10	3.9	86
20	21	10	88	172
30	15	10	181	259
40	13	10	274	345

5.2 - Relating servo commands to robot movements

Testing was conducted to relate the servo commands to the angular position of the servo. A series of pulses were sent to the servo, and the resulting servo angles were measured. The results of the testing are plotted in Figure 5-2. Based on the testing, it was assumed that each change of 10 μs in the pulses rotated the servo 1°. The measured values of the servo angle varied from the expected values by at most 10% error. This error range was considered acceptable, and the relationship of 10 μs to 1° was used throughout the remainder of the experiment.

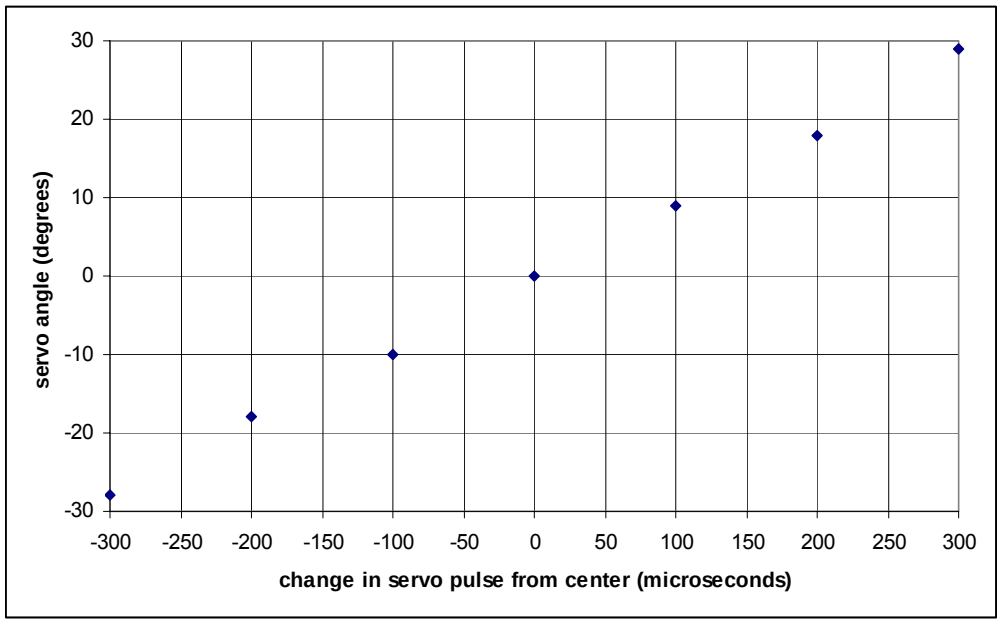


Figure 5-2. Test results comparing the servo control pulse with the resulting motion of the servo.

The control strategy applied in this research required the ability to manipulate the linear velocity and angular velocity of the robot. Given the results from the previous chapter, the step length and turning angle could be predicted. Then the robot's velocities could be calculated based on the elapsed time for the servos to complete their motion. The linear velocity of the robot, ν , was determined by the relationship

$$\nu = \frac{s}{t} \tag{5.3}$$

where s is the distance traveled during a single stepping and turning motion and t is the time required for the servos to produce this motion. Similarly, the angular velocity, ω , was calculated using

$$\omega = \frac{\theta}{t} \tag{5.4}$$

where θ stands for the turn angle produced during a single turn. Related to the total elapsed time, these velocities represented the average velocities during a complete stepping and turning motion. The velocities incorporated the time required for the roll servo and pace servo to adjust. Even though the robot would not be moving its location or turning while the roll servo turned, the time required to move the roll servo was factored into the velocities of the robot. For cases when the lead foot needed to be switched so that the desired turning direction could be executed, the time required to switch the lead foot was included in the total time for the motion.

In summary, this chapter discussed how the robot's velocities could be controlled. Given distance and angle data for each robot motion, the time elapsed during the motion was adjusted. The incremental change in servo pulses and the pause time determined the speed of the servo motors. By adjusting the servo speed, the time to complete each robot motion could be

controlled. Therefore, a method was developed to drive the servo motors in such a way to make the robot move at desired velocities.

Chapter 6 - Autonomous control of the velocities

Using the previous results as a knowledge base, a control system was designed to produce the motion expected to most closely track a trajectory. Taking the results of the robot movement and servo speed, the control system attempted to move the robot at desired velocities. This research was intended to enable the two degree-of-freedom biped to perform trajectory tracking. In designing the autonomous control, it was assumed that a higher level control translated the trajectory into discrete pairs of linear and angular velocities. The control program selected the robot movements that were expected to most closely match the desired velocities. Generated by the control algorithm, commands for the robot were input to the onboard microcontroller. Thus, the control program received a desired trajectory and output commands for the robot.

6.1 - Relating velocity to known robot movements

To achieve the desired velocities as specified by the higher level control, first the velocities were related to the robot's known movements. From before, the step length and turning angle were known. As shown previously, the robot's linear velocity, v, could be set according to

$$v = \frac{s}{t} \tag{5.3}$$

where s is the step length and t is time elapsed during the step. According to this relationship, the linear velocity is considered independent of the angular velocity. Similarly, the angular velocity, ω, could be set independently of linear velocity using the relationship

$$\omega = \frac{\theta}{t} \tag{5.4}$$

where θ represents the turning angle. Although these equations provided a way to control the linear and angular velocities independently, the trajectory tracking required that both velocities be controlled simultaneously. The higher level control generated discrete pairs of linear and angular velocities. Therefore, a method was sought to control both velocities during the same stride.

An interesting relationship between the desired velocities and the set of possible movements was found. Combining equations 5.3 and 5.4, yielded the relationship

$$\frac{\nu}{\omega} = \frac{s}{\theta} \tag{6.1}$$

This relationship shows that the ratio of linear to angular velocity could be equated to the ratio of step length to turning angle. For each pair of desired velocities, the velocity ratio could be calculated. Then the ratio of step length could be matched with the velocity ratio.

The control system autonomously selected the step length and turning angle to match the desired velocity ratio for each movement. Using the turning data that was collected for various positions of the roll servo, the ratio of step length to turning angle was calculated for each movement. From this data, lists were compiled for a Matlab function to select the ratio that most closely matched the desired velocity ratio. The function then selected the roll servo command that corresponded to the desired ratio.

6.2 - Selecting servo positions

When the initial orientation of the legs permitted turning in the desired direction, the program selected the roll command to produce the turn in a single stride. The process of selecting the roll command involved several steps. To begin, the program had to decide that the legs were oriented in the proper direction. When a clockwise turn was desired, the program checked that the left foot extended forward. For a counterclockwise turn, the program checked that the right foot was in forward. If the legs were oriented properly, the function then referred to the lists of step length and turning angles presented in Table 6-1 and Table 6-2. The lists of step length and turning angle were viewed separately by the program depending on the desired turning direction. Table 6-1 lists the data for counterclockwise turns, and Table 6-2 gives the data for clockwise turns. The table corresponding to the desired turning direction was consulted by the program. Linear interpolation was performed between the data points to find a roll command that would match the step length to turning angle ratio with the desired velocity ratio. If the magnitude of the linear velocity greatly exceeded the magnitude of the angular velocity, the motion was approximated by straight forward or reverse walking. This straight-line walking was selected when the velocity ratio exceeded the range of values provided in the selection list.

Table 6-1. Selection list for the ratio of step length to turning angle during counterclockwise turns.

s/θ	roll command	step length, s (mm)	turn angle, θ (degrees)
450	185	58	7
260	180	54	13
150	175	51	20
62	170	24	22
33	165	13	22
-45	160	-13	16
-250	155	-37	9

Table 6-2. Selection list for the ratio of step length to turning angle during clockwise turns.

s/θ	roll command	step length, s (mm)	turn angle, θ (degrees)
420	190	-43	-6
170	185	-24	-9
40	180	-8	-11
-50	175	14	-16
-175	170	32	-12
-650	165	54	-5

When the foot orientation did not allow turning in the desired direction and a step was required to switch the lead foot. A separate subroutine was used to select the roll command for the turning motion. This subroutine also contained two lists of ratios for displacement to turning angle and the corresponding roll commands. A separate list was made for each turning direction. Table 6-3 presents the selection list for counterclockwise turning, and Table 6-4 gives the data for clockwise turning. The selection lists included the displacements after the robot stepped to switch the lead foot. Turning in either direction could be preceded by a step forward or backward. The selection list for each turning direction combined the data for both stepping directions. When the lists combined data for the forward and backward steps, the linear interpolation required a special consideration. Interpolation was impossible where the two stepping directions met. Between the shortest forward and shortest backward steps, the ratio of step length to turning angle produced errors. When the velocity ratio was near zero, the matching ratio of displacement to turning angle would fall between the data for the different stepping directions. The roll command corresponding to a ratio near zero would tilt the robot far to one side or the other. A large tilt angle would lift one foot completely above the ground. With this foot free to swing, the pace motion would produce straight-line walking. However, a velocity ratio near zero would indicate that the magnitude of the angular velocity was much

greater than the magnitude of the linear velocity. Thus, the desired motion would be a predominantly turning motion, but the roll command would produce straight-line walking. To alleviate this complication, the roll command corresponding to the shortest forward step was used whenever the velocity ratio fell between the two stepping directions.

Table 6-3. Selection list for the ratio of step length to turning angle during counterclockwise turns preceded by a reorienting step.

s/θ	roll command	step length , s (mm)	turn angle, θ (degrees)
966	185	118	7
502	180	114	13
318	175	111	20
219	170	84	22
190	165	73	22
168	160	47	16
146	155	23	9
-16	185	-2	7
-26	175	-9	20
-26	180	-6	13
-94	170	-36	22
-122	165	-47	22
-261	160	-73	16
-618	155	-97	9

Table 6-4. Selection list for the ratio of step length to turning angle during clockwise turns preceded by a reorienting step.

s/θ	roll command	step length, s (mm)	turn angle, θ (degrees)
984	190	-103	-6
535	185	-84	-9
354	180	-68	-11
165	175	-46	-16
134	170	-28	-12
69	165	-6	-5
-162	190	17	-6
-229	185	36	-9
-265	175	74	-16
-271	180	52	-11
-439	170	92	-12
-1306	165	114	-5

6.3 - Adjusting servo speeds

Once the roll command was selected from the proper selection list, the displacement and turning angle of the robot was interpolated from the same list. Using the displacement and turning angle during each robot motion, the linear and angular velocities of the robot could be set to match the desired values. As shown in previous equations, the velocities could be calculated by dividing the displacement or turning angle by the time elapsed during the motion. The servos were controlled to perform the motions in the desired length of time. The parameters for the incremental change between consecutive servo pulses and the pause time between pulses were set by a Matlab subroutine. The length of time required for the servos to perform a movement was determined by considering the pulse commands. For each movement performed by the servos, the total change in the pulse commands, p, was calculated according to

$$p = |roll - atroll| + |pace - atpace| \tag{6.2}$$

where $roll$ and $pace$ are the goal commands of the roll and pace servos and $atroll$ and $atpace$ are the commands corresponding to the current servo positions. The number of pulses, n, required to move the servos through the total change in servo commands was calculated using

$$n = \frac{p}{i} \tag{6.3}$$

where i represents the incremental change between consecutive pulse commands. The period of each pulse, T, was calculated with the sum

$$T = 1.6ms + \delta \tag{6.4}$$

where δ is the pause time following each pulse. The duration of each pulse was approximated as 1.6 ms, because the roll and pace servos oscillated about their upright, standing positions.

Standing upright corresponded to pulses of length 1.70 ms for the roll servo and 1.51 ms for the pace servo. The approximated pulse length was about midway between the two pulses for standing upright.

The time required to complete a movement was found by multiplying the number of pulses by the period for each pulse according to

$$t = nT \tag{6.5}$$

which was expanded to the form

$$t = \left(\frac{p}{i}\right)(1.6ms + \delta) \tag{6.6}$$

to show the parameters for the incremental change in pulses and the pause between pulses.

With the ability to set the time required for the servos to reach their goal positions, the velocities of the robot could be controlled. For instance, the linear velocity at which the robot would move was calculated by

$$v = \frac{si}{p(1.6ms + \delta)} \tag{6.7}$$

in which the displacement has been divided by the elapsed time during the movement. The servo parameters were isolated on the right side of the equation, which resulted in

$$v\left(\frac{p}{s}\right) = \frac{i}{(1.6ms + \delta)} \tag{6.8}$$

The quotient on the right side of the equation could be considered as the rate at which the servo commands changed. A new term, $\dot{p}$, was created to represent the rate of change in the servo commands and substituted into the equation to yield

$$v\left(\frac{p}{s}\right) = \dot{p} \tag{6.9}$$

The rate of change in the servo commands related directly to the speed at which the servos turned. A list of possible combinations of the incremental pulse change and the pause between pulses was compiled. Along with these parameters, the corresponding rate of change in the servo commands was given in Table 5-1. The robot was controlled to move at the desired velocity by calculating the rate of change of servo commands and then selecting the combination of incremental pulse change and pause between pulses that most closely matched the calculated value.

After the Matlab program had selected the roll command, incremental change in servo commands, and the pause between pulses, the program needed to output the data in a format that the Stamp I could understand. The Matlab program output a list of values for the roll command, increment, and tweenpulse. For the purposes of this research, the output from the program was typed manually into the base code on the Stamp I. Further research could investigate to potential for communication between the higher level controls and the robot. The source code for the Matlab program and the microcontroller program are given in the appendix.

Feedforward control was used to show the potential for implementing feedback to achieve trajectory tracking. A Matlab program performed the feedforward control. Receiving the desired trajectory as an input, the program produced commands for the robot. Within the control algorithm, the desired trajectory was related to experimental predictions of step length and turning angle. Then, the program selected the servo speed to produce the stepping and turning at the desired velocities. Thus, the control program generated the commands that were expected to achieve the most accurate trajectory tracking.

Chapter 7 - Robot simulation

A simulation program showed the robot motion as expected by the control system. The simulation was added as an extension to the control program already created in Matlab. Extracting data from the control program provided the necessary input for the simulation. Given the kinematic solutions of the joint positions based on the tilt and stride angles, the positions of the joints were plotted in a three-dimensional graph. A series of three-dimensional plots then were combined in an animation. This animation displayed visually the expected motion of the robot. Also, the simulation created plots the expected path that the robot would follow over the ground.

7.1 - Extracting information from the control code

The simulation code visually presented information that had been computed by the control program. From the control system, the sequence of servo commands and speed parameters was output. Given a trajectory in terms of linear and angular velocities, the control algorithm matched each step to the corresponding step length and turning angle. Positioning the roll servo produced the step length and turning angle. Thus, the control system output the roll commands for the robot to implement. Paired with each roll command, pace commands were also output to generate the striding motions. To make the steps and turns occur at the desired linear and angular velocities, the control algorithm also output speed parameters. The program calculated the incremental growth of consecutive servo pulses. Coupled with the change between pulses, the pause after each pulse set the speed of the servos. The complete output from the control program included each pair of servo commands listed with the two speed parameters.

In addition to reading the control system's output, the simulation used location and heading information. Location and heading were not part of the commands given to the onboard microcontroller. However, when choosing these commands, the control algorithm had to calculate the expected step length and turning angle. The simulation program borrowed the step length and turning angle data. From this data, the simulation computed a list of the robot's location and heading after each step. The control output along with the expected location and heading of the robot supplied all the required information for the simulation.

7.2 - Visualizing the robot

To create a three-dimensional plot, the simulation constructed a stick figure of the robot from the kinematic equations. Figure 7-1 illustrates the way in which the robot was represented as a stick figure. During the animation, this stick figure walked around a three-dimensional grid as shown in Figure 7-2. Roll and pace commands were converted by the simulation into tilt and stride angles of the robot body. From the tilt and stride angles, the coordinates of the joints were solved. Initially the coordinates were defined in the robot coordinate frame, which placed the origin at the stance foot. According to the sign of the tilt angle, the program determined the direction that the robot tilted. Then the foot onto which the robot tilted was assigned the origin. The simulation plotted a line connecting the joints on the feet and the front link of each leg. Thus a stick figure was plotted in three-dimensional space with the ankle of the stance foot at the origin.

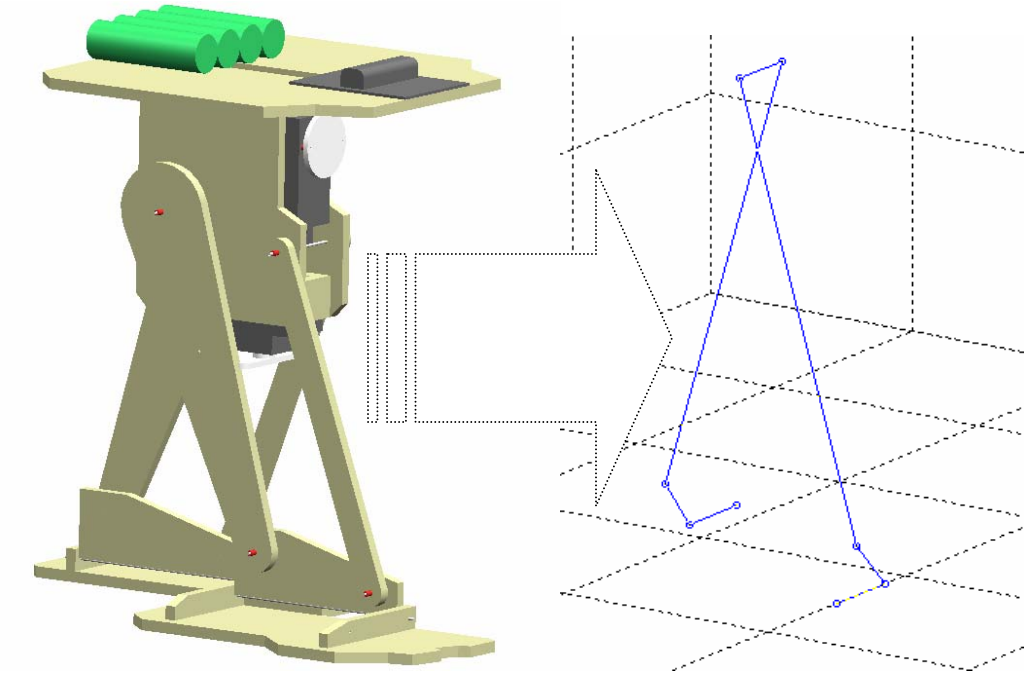


Figure 7-1. Transformation of the robot into a stick figure for the animation.

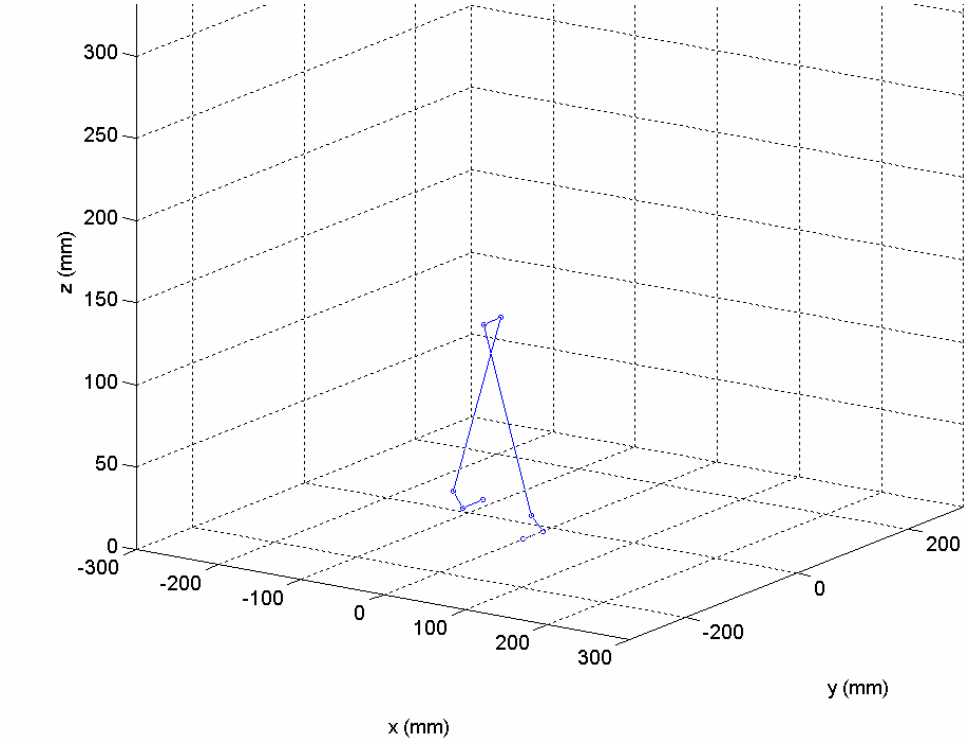


Figure 7-2. Screenshot of the animation.

After assembling a stick figure of the robot, the distance between the origins at the feet was considered. All of the robot coordinates needed to be translated to correct for this distance. A width of 27 mm separated the origins. Switching the origin from one foot to the other shifted the robot image sideways. For example, the robot could be plotted with the origin at the left foot and then replotted with the origin at the right foot. When the origin switched to the right foot, the robot would slide instantly to the left by 27 mm. During the animation, the instantaneous shifting of the image would create a discontinuous motion. Therefore, joint locations calculated relative to either ankle were translated 13.5 mm to overlap images plotted from either origin.

With the spacing between the origins corrected, the simulation moved the image to its expected location and heading. As described in Chapter 3, all joint coordinates needed to be rotated and translated to the proper position in space. To begin, the image was rotated about the vertical axis to face the direction of the heading. For this rotation, the list of coordinates was multiplied by a rotation matrix. Then the image was translated to the expected location. To translate the image, the x and y coordinates of the location were added to the joint coordinates.

7.3 - Animating the robot motion

In addition to showing the position of the robot body, the animation reflected the robot velocities. The two speed parameters for the servo motors determined the speed at which the robot would tilt and stride. Therefore, the simulation recreated the action of the servos as produced by the servo pulses. For each pulse the servos would receive, the robot image was redrawn. The joints were plotted according to the servo position that each pulse dictated. To show the speeds, the period of each servo pulse needed to elapse between movements of the

robot. The period included the duration of the pulse and the following pause. In the animation, the pause was added after plotting the effect of each pulse. Thus, each servo pulse and its following pause were represented by a redrawing of the robot followed by the same duration pause.

After animating the joints according to the servo motion, the movement of the robot through space needed to be shown. A list of expected locations and headings had been calculated for the beginning and end of each step. However, the animation produced multiple plots between the beginning and end of each step. Servo positions changed incrementally during each stepping motion, and the animation reflected these changes. Therefore, the robot's location and heading were plotted to reflect the intermediate motions. The changes in location and heading were divided into the same number of steps as the number of pulses that would be sent to the servos. As each incremental change in the servos was plotted, the animation showed the corresponding incremental change in the location and heading. A smooth motion was produced in which the robot tilted and took strides while following the desired trajectory.

One point requiring special attention was the way in which the robot location was defined. In the kinematic analysis, the robot's location had been related to the origin of the robot coordinate frame. The origin was set at the stance foot. Therefore, the stance foot was placed at the coordinates that were given as the location of the robot. When the stepping and turning movements were tested, the step length was defined as the distance from the initial lead foot to the final lead foot. At any time, the robot's location was found by combining the step lengths up to that time. The location was calculated based on the progression of the lead foot. Confusion arose when the stance foot and lead foot were not always the same.

To illustrate this confusion, a backward step can be considered. As shown in Figure 7-3, the left foot was in the initial lead position, and the right foot initially trailed behind. In this position, the robot location would have been calculated by applying the step length to find the coordinates of the lead foot. Before the backward step, the robot location would have been determined to be at the left foot. During the backward step, the right foot would have been the stance foot. Solving the joint positions according to the kinematics would relate all coordinates to the stance foot. However, the location of the stance foot was not defined.

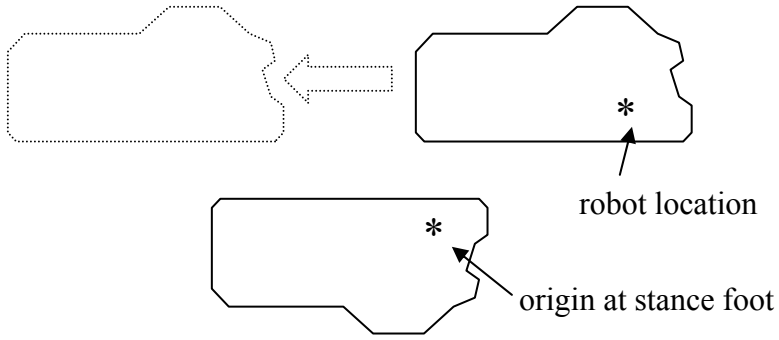


Figure 7-3. Backward stepping in which the robot location and the origin at the stance foot do not agree.

The confusion was resolved by calculating the location of both feet after every stride. Before, the location of the lead foot had been calculated. From the lead foot, the location of the trailing foot could be found. To solve the trailing foot's location, one stride length was subtracted from the lead foot's location. Knowing the location of both feet eliminated the confusion over the difference between the lead foot and stance foot. Regardless of which foot happened to be in the lead position, the location of the stance foot was known. Therefore, the robot image could be plotted relative to the stance foot, which was placed in the proper location.

7.4 - Simulating foot locations

Although the animation visually showed the robot's expected motion, quantitative data was also desired. This quantitative data was needed to make numerical comparisons between the expected and actual performance. The robot's actual performance was recorded as position data of the feet. Therefore, the simulation also output the corresponding foot positions. Once the animation was able to plot the locations of the robot joints, the coordinates of the feet could be taken directly from the animation. Also, the expected time at which the feet were in these positions was calculated. To calculate the time, the period of all preceding servo pulses was summed together. These coordinates and times were output corresponding to the instants when the actual foot locations were measured. Further explanation of the actual foot locations is provided in Chapter 8. An example of the simulated foot locations is plotted in Figure 7-4 for the robot walking forward along a counterclockwise arc.

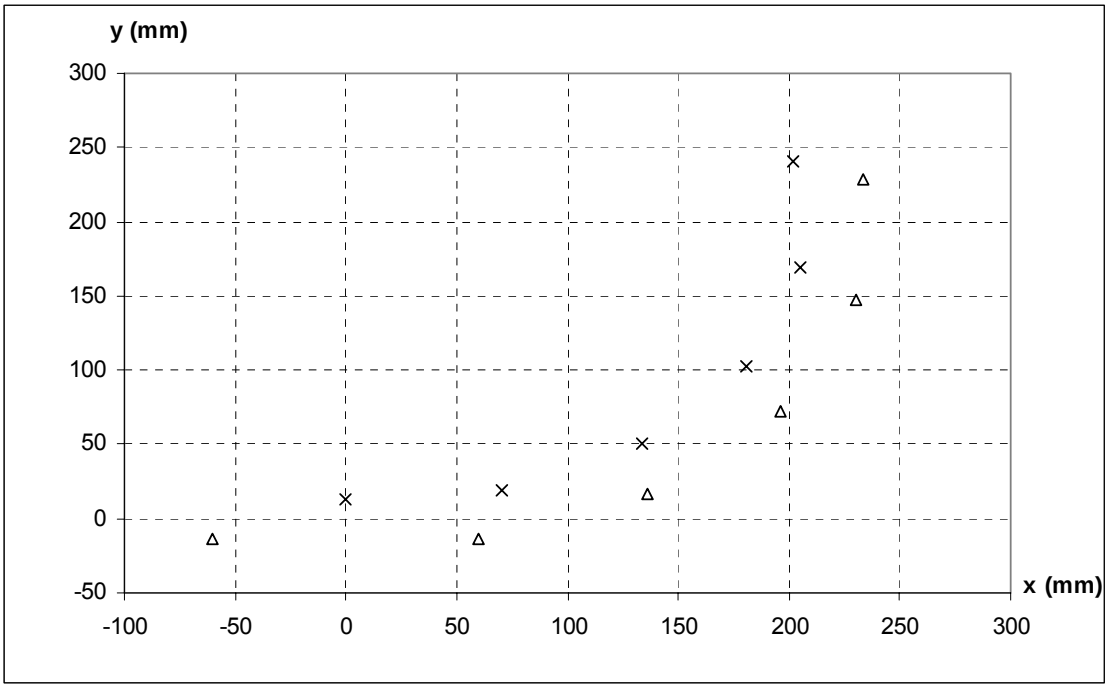


Figure 7-4. Simulated locations of the robot feet for walking forward in a counterclockwise arc. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

The simulation served as a means to compare the expected and measured performance of the robot. Viewing the animation provided a general idea of how the robot motions should look. To compare the simulation to actual measurements of the motions, location and time results were computed. In testing, the lead foot location and the time at the end of each step would be recorded. For this reason, the expected values for the location and time were output by the simulation. A plot of the expected lead foot location was created. These locations were plotted with respect to time. Also, the step lengths and turning angles were divided by the elapsed time to yield the expected linear and angular velocities. Thus, the simulation allowed the expected and actual performance of the robot to be compared visually and quantitatively.

Chapter 8 - Results and Discussion

Test results showed that the robot exhibited a wide range of error in producing the desired linear and angular velocities. Three trajectories were input individually to the control algorithm. For each trajectory, the resulting control commands were programmed to the onboard microcontroller. Then the robot attempted to track the trajectory. However, the robot drifted significantly from the desired path even after five steps. Also, the time elapsed during the movements was longer than expected. This error found in the testing suggests that the feedforward control system is inadequate. Therefore, adding a feedback mechanism is recommended to improve the robot's tracking accuracy.

For the results, it was desired that actual linear and angular velocities would be measured. These velocities could then be compared to the desired velocities input to the control program. However, no equipment was available to measure the velocities directly. Therefore, position and time data was used to infer the robot velocities.

To determine the robot position, marks were made at the location of the feet. These marks were drawn manually on the walking surface at the front instep corner of each foot. Foot locations were recorded only at certain times. When the robot moved with a positive linear velocity, the lead foot was marked each time it reached its full forward extension. If the robot walked with a negative linear velocity, the trailing foot was marked each time it reached its full backward extension. As the feet switched trailing and leading orientations, a series of alternating locations for the left and right foot was marked.

After the robot finished walking, the marks were measured and recorded in terms of Cartesian coordinates. The origin of the Cartesian reference frame was set relative to the initial

position of the robot. For each test, the robot began with the left foot extended forward. A mark at the left foot established the origin point of the reference frame. The orientation of the reference frame was defined by aligning the x-axis with the direction that the robot initially faced. According to this reference frame, an x-position and y-position was measured for each mark of the foot locations.

Three tests were run with the robot. First the robot was programmed to walk forward in a straight line at 20 mm/s. Then the robot attempted to track a trajectory with a constant linear velocity of 10 mm/s and angular velocity of 0.05 radians/s. This second pattern caused the robot to walk forward in a counterclockwise arc. For the final test, a desired trajectory was input with a constant linear velocity of -10 mm/s and angular velocity of -0.05 radians/s. The final pattern made the robot walk in reverse along a clockwise arc.

8.1 - Walking forward in a straight line

For forward walking in a straight line, the expected and actual foot locations agreed more closely than in the other two tests. Plots of the foot locations are plotted in Figure 8-1, which shows the simulated data, and Figure 8-2, which presents the experimental measurements. Straight line walking avoided many of the sources of error. To take each step, the robot lifted the swing leg above the ground and moved the leg forward. Raising the leg eliminated sliding contacts with the ground. Thus, irregularities in the walking surface could not affect the walking motions. Slight differences between the expected and actual foot locations may have arisen from the loose joints. The joint construction allowed the robot to bend and stretch unpredictably. Possibly the width between the feet changed from one step to the next. Also, the bending of the robot caused the swing foot to touch the ground a shortly before each step was complete. When

the swing foot touched down, the sliding friction may have shortened the step length or caused slight turning. The possibility of shortened step lengths is supported by the results. After five steps, the actual position of the robot's right foot fell short by about eight millimeters in the x-direction or about three percent error.

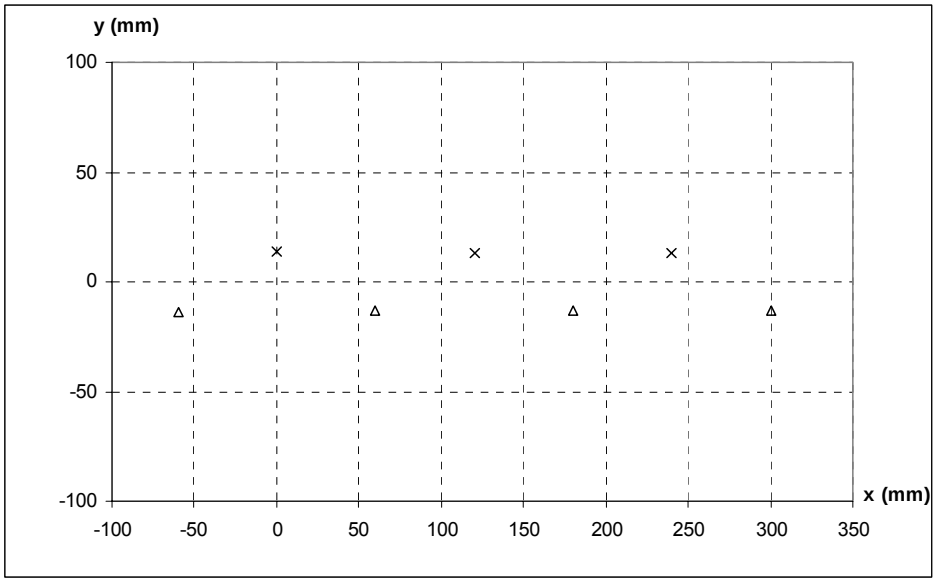


Figure 8-1. Expected foot locations for walking forward in a straight line. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

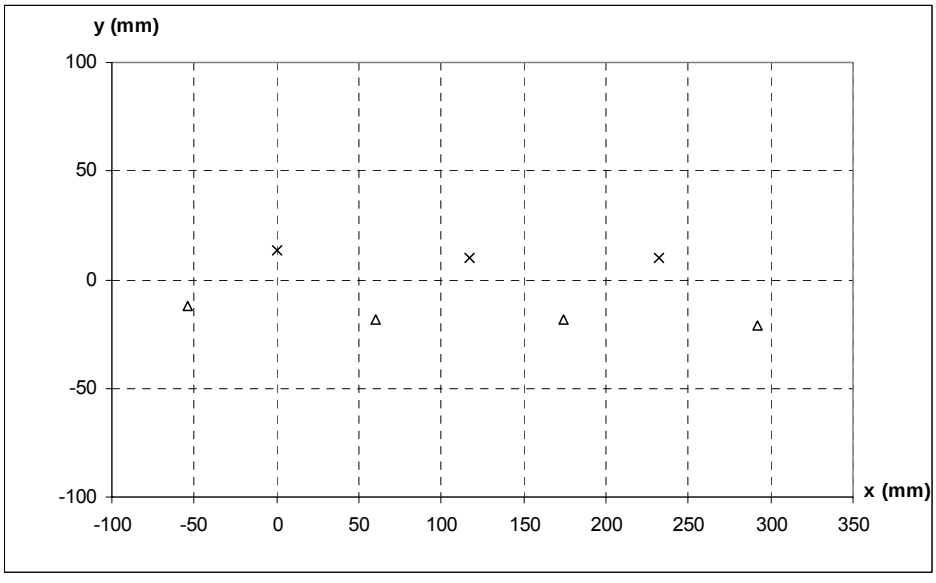


Figure 8-2. Actual foot locations for walking forward in a straight line. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

Although the actual foot locations agreed well with expected values, the linear velocity showed considerable error. The performance data including calculated velocities is listed in Table 8-1. Velocity was calculated by dividing the distance a foot traveled by the elapsed time. Left and right foot locations were considered separately in calculating the velocity. For example, the displacement of the left foot was defined as the distance between consecutive locations. Dividing this displacement by the elapsed time between these two points yielded the linear velocity. The same calculations were performed with respect to the right foot.

Table 8-1. Actual performance data for walking forward in a straight line.

time (s)	left foot displacement (mm)	right foot displacement (mm)	velocity (mm/s)	error (%)
0	0	0		
3.4		115		
7.2	118		16	20
10.7		114	16	20
14.4	114		16	20
18.1		118	16	20

Compared to the desired velocity of 20 mm/s, the actual velocities fell short consistently by 4 mm/s. Although the actual velocities were inaccurate by about twenty percent, the values were precise. This precision implies that a scaling factor could correct for the velocity. In the control algorithm, the desired velocity could be increased by twenty percent. Then the corresponding output would be expected to produce a velocity much closer to the desired value.

Partly, the shorter step lengths contributed to slower linear velocities. Additionally, the time elapsed during the robot's movements exceeded the expected time. Table 8-2 presents the expected time data from the simulation and the actual times measured during the test. Because

the robot took longer to perform each step, the average velocity decreased. Each step actually lasted up to eighteen percent longer than the simulation indicated. These longer elapsed times probably were caused by delays in the microcontroller. The control program did not account for processing time. In the control program, elapsed times were calculated based on the servo pulse duration and pauses between pulses. Because each pause lasted at least ten milliseconds, the processing time was considered negligible. However, processing time accumulated as the robot operated, and the robot moved significantly slower than desired. Additional work could take processing time into account when controlling the robot.

Table 8-2. Expected and actual time data.

expected time (s)	actual time (s)	difference (%)
0	0	0
3.1	3.4	11
6.1	7.2	17
9.2	10.7	16
12.3	14.4	18
15.3	18.1	18

8.2 - Walking forward in a counterclockwise arc

The second trajectory that the robot attempted to track involved a forward, counterclockwise path. While following this path, the robot was intended to move with a linear velocity of 10 mm/s and an angular velocity of 0.05 radians/s. With turning added, the second test revealed additional error that was not evident in straight line walking. Linear velocity error increased, and less precision was evident in the actual velocities. Also, the angular velocity varied widely from the expected values. Plots of the simulated footfalls and the actual locations of the feet are displayed in Figure 8-3 and Figure 8-4.

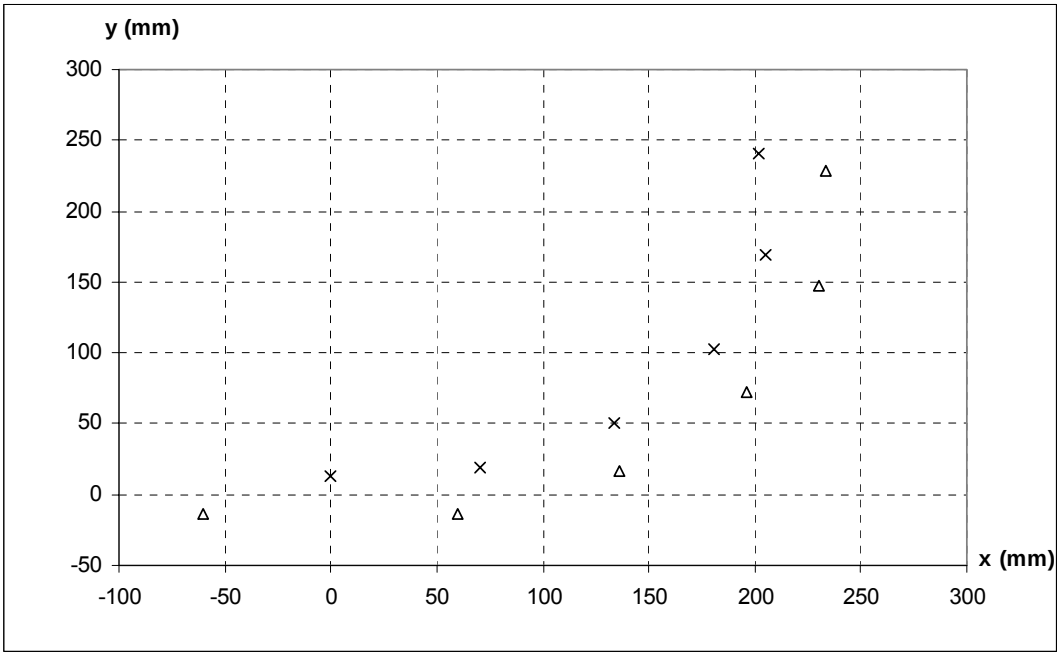


Figure 8-3. Expected foot locations for walking forward in a counterclockwise arc. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

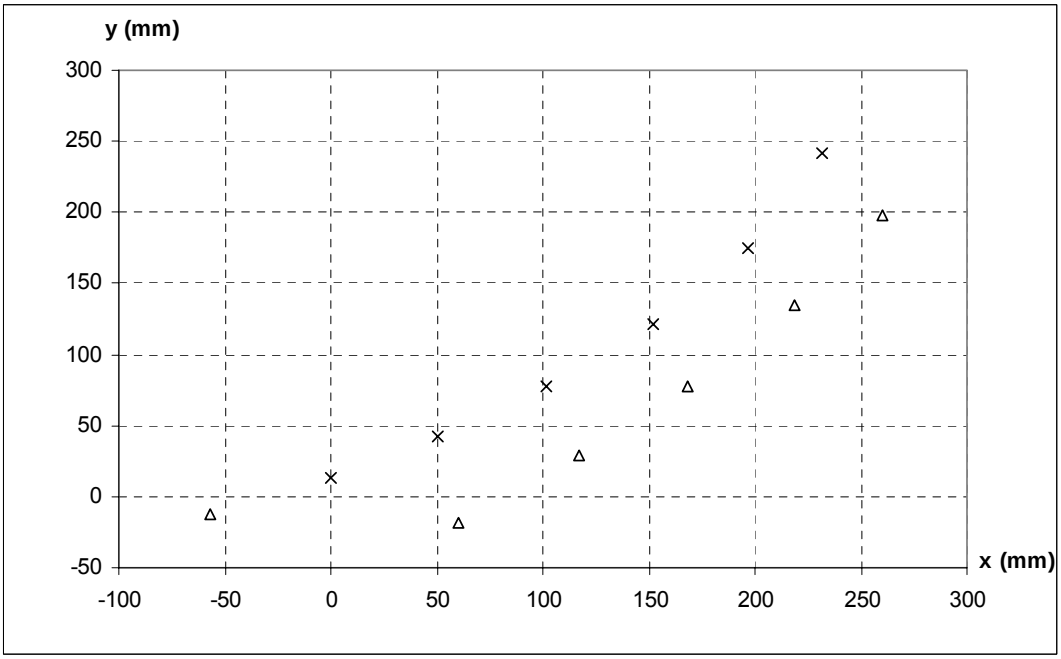


Figure 8-4. Actual foot locations for walking forward in a counterclockwise arc. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

As in the previous test, velocity was calculated by dividing the displacement of one foot by the elapsed time. To simplify the analysis for the second test, only the left foot was considered. During each movement, the right foot lifted and swung forward into the lead position. Then both feet dragged along the ground to produce counterclockwise turning. The right foot lifted for each step, but the left foot remained in contact with the ground during the entire test. Therefore, the left foot moved along a continuous path from one recorded location to the next. It was assumed that the left foot provided a better representation of the robot's average velocities.

Calculated values for the displacement and linear velocity are listed in Table 8-3. As shown in the table, the error range varied from nine to twenty-nine percent slower than the desired velocity of 10 mm/s. According to the simulation, the left foot was expected to move about 71 mm between data points. However, all but one of the actual displacements fell short of the expected distance. These shorter displacements decreased the velocity. Again, the actual time exceeded the expected time for each motion. Expected and measured times for each footfall are given in Table 8-4. The motions may have been delayed by processing time in the microcontroller. In the first test, the actual times were about eighteen percent longer than the simulated times. For the second test, most times were about thirteen percent longer than expected. The second test probably produced more accurate times, because less tilting was required during each motion. Therefore, less processing time accumulated during each repositioning of the roll servo. Still, the actual times were longer than expected, which contributed to the slower velocities.

Table 8-3. Actual linear velocity for walking forward in a counterclockwise arc.

time (s)	displacement (mm)	velocity (mm/s)	error (%)
0.0	0		
8.2	58	7.1	-29
17.2	62	6.8	-32
25.8	68	7.8	-22
34.3	70	8.2	-18
42.6	75	9.1	-9

Table 8-4. Expected and actual time data.

expected time (s)	actual time (s)	difference (%)
0	0	0
3.9	4.4	13
7.7	8.2	6
11.5	12.9	12
15.3	17.2	12
19.1	21.7	14
22.9	25.8	13
26.7	30.3	13
30.5	34.3	12
34.3	38.7	13
38.1	42.6	12

Angular velocity also was inferred from the position and time data. The robot heading was calculated by finding the direction to move between consecutive foot locations. Each change in heading was determined to be the turning angle. By dividing the turning angle by the elapsed time, angular velocity was calculated. Table 8-5 lists calculated values for the angular velocity.

The angular velocity varied with large error from the desired velocity. During the first turn, the robot exceeded the desired angular velocity of 0.05 radians/s. However, all other turns were slower than the desired velocity by fifty to eighty percent. Possibly, variations in the walking surface contributed to the error. The robot walked on a tabletop that sagged slightly in

the middle. A slope caused by the sagging may have influenced the weight distribution between the feet. Changing the weight distribution would have affected the turning angle. Also, particles on the walking surface may have restricted the turning motion. The large variation in the angular velocity certainly makes feedforward control unacceptable for controlling the robot's turning.

Table 8-5. Actual angular velocity for walking forward in a counterclockwise arc.

time (s)	turning angle (radians)	angular velocity (radians/s)	error (%)
0.0	0		
8.2	0.51	0.06	25
17.2	0.09	0.01	-80
25.8	0.12	0.01	-73
34.3	0.16	0.02	-62
42.6	0.21	0.02	-50

8.3 - Walking backward in a clockwise arc

For the final test, the robot attempted to follow a trajectory with a constant -10 mm/s and -0.05 radians/s. The robot moved in the opposite direction for this test than for the previous test. Figure 8-5 and Figure 8-6 show the expected footfalls from the simulation and the measured data from the final test. The visual appearances of the two plots are noticeably different. During the test, the robot moved farther in the negative x-direction while turning less than the simulation predicted.

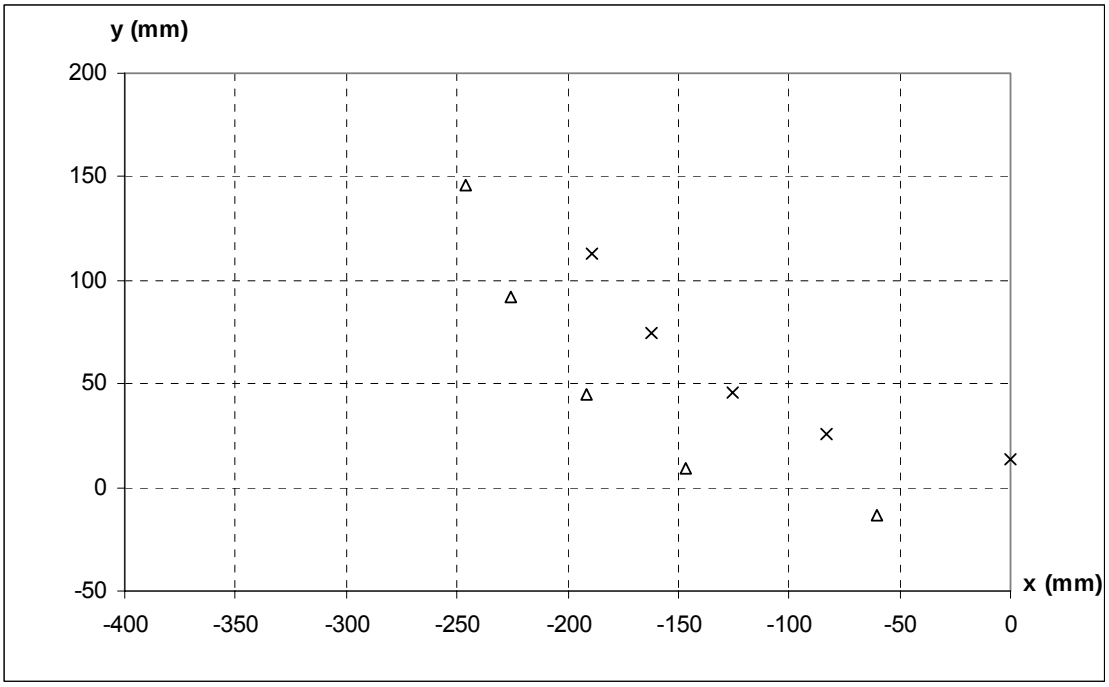


Figure 8-5. Expected foot locations for walking backward in a clockwise arc. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

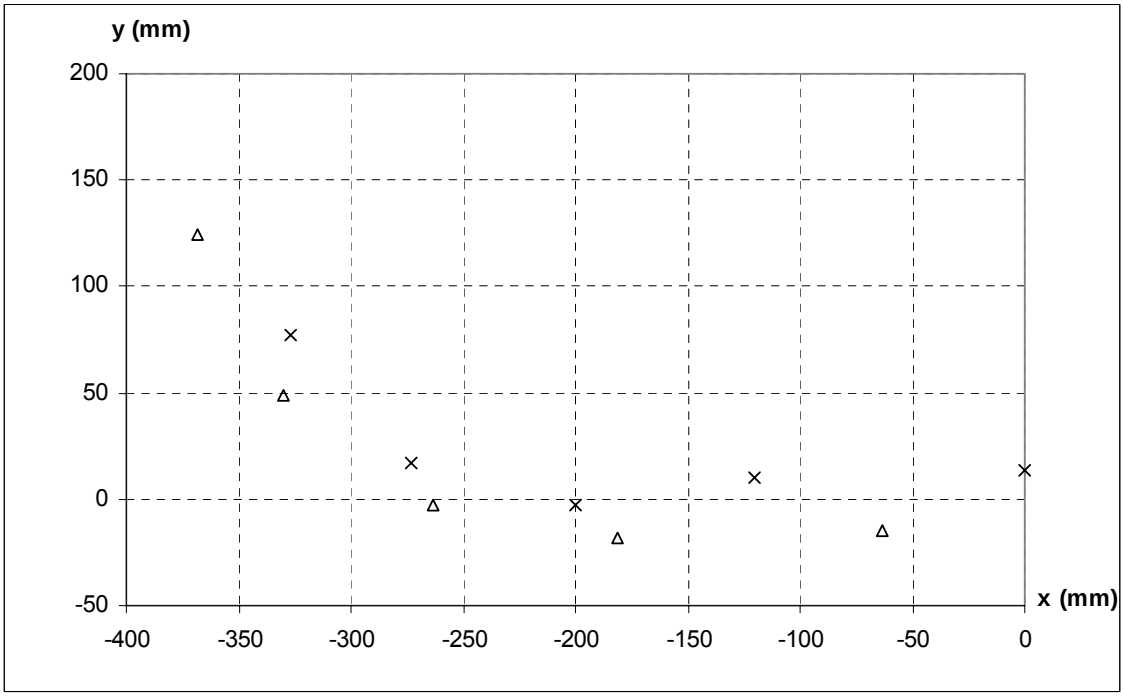


Figure 8-6. Actual foot locations for walking backward in a clockwise arc. Each x marks a position of the left foot. Each triangle marks a position of the right foot.

Linear and angular velocities were calculated in the same way as in the previous test. Calculated values for the displacement and linear velocity are listed in Table 8-6. The recorded times used in the velocity calculations are tabulated with the expected times in Table 8-7. Although the actual times again exceeded the expected times, the robot moved faster. During the test, the robot moved with greater speed in the negative direction than the desired velocity. Calculating the displacement between consecutive footfalls in the simulation revealed a predicted displacement of about -47 mm. However, the actual displacements exceeded the predicted values by at least sixty percent. Longer displacements would suggest that the robot walked more in a straight line and turned less. The actual path of the robot did have less curvature overall than the simulated path.

Table 8-6. Actual linear velocity for walking backward in a clockwise arc.

time (s)	displacement (mm)	velocity (mm/s)	error (%)
0	0		
3.5	-121		
9.5	-80	-13	-34
15.7	-75	-12	-22
22.0	-81	-13	-28

Table 8-7. Expected and actual time data.

expected time (s)	actual time (s)	difference (%)
0	0	0
3.0	3.5	18
5.6	6.8	22
8.1	9.5	18
10.7	12.7	19
13.3	15.7	18
15.9	18.8	18
18.5	22.0	19
21.1	25.2	20

Results of the angular velocity showed the largest range of error for all the tests. Table 8-2 lists the inferred values of the turning angle. At the beginning of the test, the robot even turned in the opposite direction from the desired motion. As the robot continued to move, the turning speed exceeded the desired value. One possible explanation for this erratic behavior is that the robot's joints had loosened by the time the third test was run. Before running the three tests, the tubing at each joint was secured with glue. However, stresses during walking may have broken some of the glue bonds. If this were the case, the turn in the wrong direction could have been caused by a misrepresentation of the data. What looked like a turn may have been the left foot sliding closer to the right foot to decrease the width separating the feet. If the hip joints were loose, the legs could pinch inward. Conversely, the increased turning angle observed towards the end of the test may have been the left leg spreading outward again. Also, irregularities in the walking surface could have contributed to the turning error.

Table 8-8. Actual angular velocity for walking backward in a clockwise arc.

time (s)	turning angle (radians)	angular velocity (radians/s)	error (%)
0	0		
3.5	0.03		
9.5	0.13	0.02	144
15.7	-0.39	-0.06	-25
22.0	-0.45	-0.07	-44

Testing three attempts at trajectory tracking showed that feedforward control was inadequate. Because the linear and angular velocities exhibited a large range of error, a feedback mechanism would be necessary to increase the robot's accuracy. Straight line walking produced

the most accurate results. Modifying the control system to account for processing time in the microcontroller could improve straight line walking. In the tests that combined walking and turning, the both linear and angular velocities varied from the desired values. Over time, the robot could stray far from the desired trajectory. Therefore, it is recommended that feedback be included in the control system. After each step the desired velocities could be adjusted to draw the robot back to the desired trajectory.

Chapter 9 - Conclusions

9.1 - Review of the thesis

This thesis has attempted to describe in detail the research conducted with a two degree-of-freedom biped. First the robot construction was considered. The robot design was chosen for its low cost and simplicity of operation. However, the construction did include loose joints that may have been a source of error in the experiments. The kinematic analysis was explained to show how all joint locations were solved. Although the control algorithm did not use the kinematics, calculating the joint locations allowed the robot motions to be predicted in a simulation. In the simulation, the kinematic equations were used to plot expected footfalls and to generate a three-dimensional animation.

After discussing the construction and kinematics, this report walked through the experimental process of developing the control system. Studying the robot movements revealed patterns in the step length and turning angle that the robot could produce. A list of possible movements was created to relate the resulting motion to various positions of the roll servo. After finding a way to adjust the step length and turning angle, control over the robot's velocities was investigated. By setting the servo speed, the linear and angular velocities of the robot were influenced. The servo speed was commanded so that the correct time would elapse during each movement. Ensuring that the desired location and orientation would be reached at the correct time would produce the desired velocities.

Once a method had been found to manipulate the velocities, a control program was written to produce the desired velocities autonomously. From a list of possible robot movements, the control program interpolated the best movement to match the desired path.

Parameters for the servo speed were selected to perform the movement with the desired velocities. The performance of the control system was tested using a simulation and prototype. In the simulation, a stick figure of the robot was animated to show the expected walking motions. Tests were performed with the prototype, and the results were compared with the simulation. It was found that a large degree of error was present in the linear and angular velocities. Therefore, feedback control is recommended to improve the accuracy of the robot's trajectory tracking.

9.2 - Evaluation of the research

This investigation demonstrated the potential of using a simple walker to perform trajectory tracking. Having only two degrees of freedom, the biped could tilt to the side and swing its legs about the hip joints. A method was developed to combine tilting and striding motions to adjust the robot's linear and angular velocities. Feedforward control was implemented to test the robot's ability to track a given trajectory. Large errors showed that a feedback mechanism would be necessary to improve the tracking accuracy. Limitations to the robot's construction and method of turning were considered. The construction was sufficient for this research, but more durability would be required for use over extended times. Construction changes and expansion of the control system is recommended for future improvements to the robot.

This research succeeded in creating a control method to make the robot respond to an input of linear and angular velocities. The control system selected commands for the onboard microcontroller to produce a motion similar to the desired trajectory. Trends in the desired trajectory were followed. When the desired velocity increased, the robot was controlled to move faster. Upon receiving angular velocity input, the control system caused to robot to turn in the

proper direction. Although the actual velocities were observed to vary from the desired velocities, the current control system algorithm could serve as part of a feedback control system. A higher level controller could adjust the desired velocities according to feedback from the robot's actual motion. The current controller would then respond to the adjusted velocity and change the robot motion.

In this research, a simple robot construction was desired for several reasons. One main attraction to this robot was its low cost. Purchasing the Bigfoot robot kit minimized expenses for this project. Also, the design contained few parts and could be assembled easily. At the beginning of this project, the robot was built and running within a short time. Because the robot was easy to assemble, repairs and modifications were made quickly using common tools. This type of simple design would be ideal for users without background knowledge in robotics. Robotics expertise would not be essential for a person to operate and maintain this robot.

A scenario can be imagined in which robots would be used for demining in developing countries. These robots could take the place of human deminers in this dangerous occupation. If the robots were easy to operate, the human deminers could become the robots' operators. The people would not need extensive robotics training. Also, inexpensive robots would be less costly to replace in case they were destroyed by mines. On the other hand, a harsh outdoor environment could increase the probability that the robots would incur damage. Having been damaged, a simple robot could be repaired quickly and easily. A robot based on the biped walker studied in this research could be an effective platform for demining applications.

As the robot is designed currently, several limitations are evident. The construction materials limited the durability of the robot. Occasionally, the wire joints loosened, and the flexibility of the robot became unpredictable. After operating the robot a long time, the joints

potentially could detach. The balsa wood parts could not endure outdoor conditions. Rain or excess dampness could warp the wooden members. Another limitation to using wooden links is that the holes for the wire joints would enlarge over time. Larger holes for the wire would further increase the looseness of the joints.

Other limitations may be inherent in the turning method. Tests with this robot were performed only on a relatively flat, even surface. It was suggested that irregularities in the walking surface may have contributed to the robot's tracking error. A slight slope in the walking surface or particles and scratches may have influenced the walking motions. In outdoor applications, the terrain could be expected to have much larger irregularities than the tested walking surface. The turning method may not be reliable on natural terrain. However, modifications to the robot construction and control system may increase the terrain adaptability of a similar biped.

Expanding on the current control strategy and improving the robot construction could lead to an effective demining robot. Because the feedback control was able to respond to trends in the desired trajectory, this control method could be included in a larger feedback system. Also, changes to the robot construction could improve the capabilities of the robot. Thus, the robot studied in this research could serve as the basis of an improved robot design.

9.3 - Recommendations for future work

Investigation into modifying the robot used in this research is recommended for future work. A scenario has been considered in which a similar type of biped could be used in demining applications. Feedback control would need to be added before the robot could operate effectively in any autonomous application. Also, physical design changes could enable the robot

to work on natural terrain. Currently, no statically-stable bipeds with a turning motion produced by scuffing the feet are known to perform useful work. However, the potential of such a design to achieve trajectory tracking has been highlighted in this research. In the future, similar robots may be developed and used for important tasks.

Ways in which the robot could improve have been revealed in the experiment. In keeping with the demining scenario, changes were considered that would make the robot a potential platform for demining. Improvements are recommended according to the following list in order of priority:

1. feedback control system
    a. camera for position feedback
    b. algorithm for adjusting desired velocities
    c. autonomous communication between devices
2. more durable parts
    a. plastic parts to replace balsa wood
    b. nylon bearings for joints
    c. enclosed compartments for electronics
3. increased size of robot
    a. larger torso with mount for mine sensor
    b. wider feet for increased range of statically stable motion
    c. longer legs for stepping across terrain discontinuities
4. extendable legs for stepping over obstacles
5. remodeled shape of feet for more uniform turning

6. added actuators for turning
7. dynamic stability included in the control

Following this list could lead to an improved robot. Because no similar robot exists, new problems and questions certainly would be encountered. The most effective improvements probably would vary from the list provided here.

Feedback control is listed as the highest priority, because the tracking error must be corrected before the robot can operate reliably. Testing demonstrated significant error levels for the linear and angular velocities. Even after a few steps, the path of the robot was observed to diverge noticeably from a desired trajectory. Feedback could be used to adjust the desired linear and angular velocities input to the current control algorithm. A camera could monitor the area in which the robot operated. Using image processing, a certain point on the robot could be identified. The location of this point could be translated into Cartesian coordinates. Then, an observer function could determine the linear and angular velocity of the robot based on changes in the coordinates. Autonomous communication between the camera, the control algorithm, and the microcontroller on the robot could complete the control loop.

Constructing the robot in a more durable way is recommended after the control network is available. Using plastic for the body parts would make the robot more weather resistant. Nylon bearings could be included to strengthen the joints. These bearings could eliminate any error caused by the looseness of the wire joints. Also, enclosing the electrical components would protect these devices from dirt and corrosion.

Increasing the size of the robot could provide a better demining platform. A larger torso could include a mount for mine detecting sensors. Widening the feet would provide a larger

stable area over which the center of gravity could be moved. Allowing the center of gravity to move more would increase the range of motion for the robot. By lengthening the legs, the robot would have more ability to step over depressions or crevices in the terrain.

Extendable legs were considered for increasing the robot's terrain adaptability. The legs could retract to step on or over obstacles. Adding extendable legs would increase the complexity of the control. More actuators would need to be positioned, and stability could become difficult. By increasing the number of components, the robot could become more difficult to build and maintain. To determine the value of extendable legs, the tradeoff between increased abilities and increased complexity would need to be investigated.

Other improvements could include reshaping the feet, adding actuators for turning, and expanding the control to permit dynamically stable motions. FEA analysis could be performed on the feet to find a better design. The feet could be reshaped to minimize the effect of terrain irregularities on the turning motion. Possibly, the front and rear edges of the feet could be curved to allow the feet to slide over small bumps. Different materials could be tested for the underside of the feet to alter the sliding friction with the ground. An alternative method to improving the turning ability of the robot would be to add actuators for turning. These actuators could increase the degrees of freedom to include a rotation in the ground plane. Finally, the robot's performance could be enhanced by using dynamically stable motions. Dynamically stable gaits could allow the robot to move faster by running. Also, the robot could be made to jump over obstacles.

The robot used in this research has the potential to be transformed into a fully autonomous work robot. Recommendations have been made to develop a demining robot. This research has shown the potential of using feedback control to enable a simplistic robot to

perform trajectory tracking. Adding to this accomplishment, a biped robot could be designed to maneuver over natural terrain. By continuing to limit the robot's complexity, the new design would be accessible to operators who are untrained in robotics. Also, expenses could be conserved by maintaining a relatively simple design. Such an accessible, low cost robot would be ideal for demining applications in developing countries.

References

[1] K. Kato and S. Hirose. *Proposition of the Humanitarian Demining System by the Quadruped Walking Robot*. IEEE Int. Conf. on Intelligent Robots and Systems. 2000.

[2] M. Raibert. *Legged robots that balance*. MIT Press, 1986.

[3] C. Shih. *Ascending and Descending Stairs for a Biped Robot*. IEEE Transactions on Systems, Man, and Cybernetics – Part A. Vol. 29, No. 3, May 1999.

[4] C. Shih and C. Chiou. *The Motion Control of a Statically Stable Biped Robot on an Uneven Floor*. IEEE Transactions on Systems, Man, and Cybernetics – Part B. Vol. 28, No. 2, April 1998.

[5] M. Kumagai and T. Emura. *Sensor-Based Walking of Human Type Biped Robot That Has 14 Degree of Freedoms*. IEEE 1997.

[6] M. Yagi. *Biped robot locomotion in scenes with unknown obstacles*. Dissertation. The University of Wisonsin, Madison, 2000.

[7] S. Kajita and K. Tani. *Adaptive Gait Control of a Biped Robot based on Realtime Sensing of the Ground Profile*. IEEE Int. Conf. on Robotics and Automation. April 1996.

[8] H. Kimura, Y. Fukuoka, K. Konaga, Y. Hada, and K. Takase. *Towards 3D Adaptive Dynamic Walking of a Quadruped Robot on Irregular Terrain by Using Neural System Model*. IEEE Int. Conf. on Intelligent Robots and Systems. October-November 2001.

[9] Y. Hong, H. Lee, S. Yi, and C. Lee. *The design and control of a jointed-leg type of a quadrupedal robot for locomotion on irregular ground*. Robotica. Vol. 17. 1999.

[10] T. McGeer. *Dynamics and Control of Bipedal Locomotion*. J. Theoretical Biology. Vol. 163. 1993.

[11] T. McGeer. *Passive Walking with Knees*. IEEE 1990.

[12] M. Coleman, M. Garcia, K. Mombaur, and A. Ruina. *Prediction of stable walking for a toy that cannot stand*. Physical Review. Vol. 64. July 2001.

[13] M. Coleman and A. Ruina. *An Uncontrolled Toy That Can Walk But Cannot Stand Still*. Physical Review. November 1997.

[14] A. Kuo. *Stabilization of Lateral Motion in Passive Dynamic Walking*. Int. J. of Robotics Research. Vol. 18, No. 9, September 1999.

[15] S. Collins, M. Wisse, and A. Ruina. *A Three-Dimensional Passive-Dynamic Walking Robot with Two Legs and Knees. Int. J. of Robotics Research*. Vol. 20, No. 7, July 2001.

[16] M. Wisse, A. Schwab, R. van der Linde. *A 3D passive dynamic biped with yaw and roll compensation*. Robotica. Vol. 19. 2001.

[17] H. Dankowicz, J. Adolfsson, and A. Nordmark. *Repetitive Gait of Passive Bipedal Mechanisms in a Three-Dimensional Environment*. Trans. of the ASME. Vol. 123. February 2001.

[18] A. Kuo. *Energetics of Actively Powered Locomotion Using the Simplest Walking Model*. ASME J. of Biomechanical Engineering. Vol. 124. February 2002.

[19] R. van der Linde. *Active leg compliance for passive walking*. IEEE Int. Conf. on Robotics and Automation. May 1998.

[20] E. Raby and D. Orin. *Passive Walking with Leg Compliance for Energy Efficient Multilegged Vehicles*. IEEE Int. Conf. on Robotics and Automation. May 1999.

[21] J. Stulce. *Conceptual Design and Simulation of a Multibody Passive-Legged Crawling Vehicle*. Dissertation. Virginia Polytechnic Institute. 2002.

[22] A. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer: London. 2000.

Appendix A: Matlab feedforward control program

```
%Main biped robot program
%This program uses a list of linear and angular velocities and selects the
%robot commands to produce these desired velocities.  The desired
%velocities are input at the beginning of the main program.  Then the
%servo_position subroutine is selected to calculate the best commands for
%the roll and pace servos on the robot.  Lastly main program assembles the
%calculated data into a form that can be plotted in the simulation.
```

```
clear all
close all

%input matrix of velocity terms describing the desired trajectory
%linear velocity in the first row
%angular velocity in the second row
%several example matrices are listed here

vw = [10 10 10 10 10 10 10;
      .05 .05 .05 -.05 -.05 -.05 -.05];
%vw = [20 20 20 20 20 20;
%      0 0 0 0 0 0];
%vw = [-20 -20 -20 -20 -20 -20;
%    0 0 0 0 0 0];
%vw = [10 10 10 10 10 10;
%    .05 .05 .05 .05 .05 .05];
%vw = [-10 -10 -10 -10 -10;
%    .05 .05 .05 .05 .05];
%vw = [10 10 10 10 10;
%    -.05 -.05 -.05 -.05 -.05];
%vw = [-10 -10 -10 -10 -10 -10;
%    -.05 -.05 -.05 -.05 -.05 -.05];
%vw = [-10 -10 -10 10 10 10;
%    -.05 -.05 -.05 .05 .05 .05];

%repeat last vw pair to create the proper motion on the last step
vw = [vw(1,:) vw(1,size(vw,2)); vw(2,:) vw(2,size(vw,2))];

r_stand = 170;
p_stand = 151;
p_left_fd = 96;
p_right_fd = 206;
atroll = 170;
atpace = 96;


%call servo_position to find the best commands for the roll and pace
%servos, the expected step length of the resulting motion, the expected
%turning angle of the motion, and the best settings of the speed parameters
%increment and tweenpulse.
[roll_pace, steplength, turn_angle, inc, tween] = servo_position(vw);
%output the desired increment, tweenpulse, roll command, pace command
%(these values may be entered by hand into the Stamp program)
inc
tween
roll_pace

%build a list of hip and tilt angles for plotting
heading(1,1) = 0;
left_location(1,1:2) = [0 0];
```

```
right_location(1,1:2) = [-60 0];
j=1;
for i=1:size(roll_pace,1)
    if roll_pace(i,1) == 0           %if no step required to switch lead foot
        plotter(j,1) = roll_pace(i,3);          %roll command
        plotter(j,2) = roll_pace(i,4);          %pace command

        s(j,1) = steplength(i,1);               %finding expected step length
                                                %and turning angle
        theta(j,1) = turn_angle(i,1);
        heading(j+1,1) = heading(j,1) + theta(j,1);

        if plotter(j,2) == p_right_fd     %left foot was previous lead foot
           right_location(j+1,:) = left_location(j,:)…
             + [s(j,1)*cos(heading(j+1,1)) s(j,1)*sin(heading(j+1,1))];
           left_location(j+1,:) = right_location(j+1,:)…
             - [60*cos(heading(j+1,1)) 60*sin(heading(j+1,1))];
        else                              %right foot was previous lead foot
           left_location(j+1,:) = right_location(j,:)…
             + [s(j,1)*cos(heading(j+1,1)) s(j,1)*sin(heading(j+1,1))];
           right_location(j+1,:) = left_location(j+1,:)…
             - [60*cos(heading(j+1,1)) 60*sin(heading(j+1,1))];
        end

        increment(j,1) = inc(i,1); %incremental change between servo commands
        tweenpulse(j,1) = tween(i,1);            %pause between servo pusles
        j = j+1;

    else                              %a step is required to switch lead foot
        plotter(j,1) = roll_pace(i,1);           %roll command to switch feet
        plotter(j,2) = roll_pace(i,2);           %pace command to switch feet

        if steplength > 0     %finding expected step length and turning angle
        s(j,1) = 60;
        else
        s(j,1) = -60;
        end
        theta(j,1) = 0;
        heading(j+1,1) = heading(j,1);

        if plotter(j,2) == p_right_fd     %left foot was previous lead foot
           right_location(j+1,:) = left_location(j,:)…
             + [s(j,1)*cos(heading(j+1,1)) s(j,1)*sin(heading(j+1,1))];
           left_location(j+1,:) = right_location(j+1,:)…
             - [60*cos(heading(j+1,1)) 60*sin(heading(j+1,1))];
        else                              %right foot was previous lead foot
           left_location(j+1,:) = right_location(j,:)…
             + [s(j,1)*cos(heading(j+1,1)) s(j,1)*sin(heading(j+1,1))];
           right_location(j+1,:) = left_location(j+1,:)…
             - [60*cos(heading(j+1,1)) 60*sin(heading(j+1,1))];
        end

        increment(j,1) = inc(i,1); %incremental change between servo commands
        tweenpulse(j,1) = tween(i,1);            %pause between servo pusles
        j = j+1;

        plotter(j,1) = roll_pace(i,3);           %roll command
        plotter(j,2) = roll_pace(i,4);           %pace command

        if steplength > 0     %finding expected step length and turning angle
        s(j,1) = steplength(i,1)-60;
        else
        s(j,1) = steplength(i,1)+60;
        end
```

```
        theta(j,1) = turn_angle(i,1);
        heading(j+1,1) = heading(j,1) + theta(j,1);

        if plotter(j,2) == p_right_fd     %left foot was previous lead foot
           right_location(j+1,:) = left_location(j,:)…
             + [s(j,1)*cos(heading(j+1,1)) s(j,1)*sin(heading(j+1,1))];
           left_location(j+1,:) = right_location(j+1,:)…
             - [60*cos(heading(j+1,1)) 60*sin(heading(j+1,1))];
        else                              %right foot was previous lead foot
           left_location(j+1,:) = right_location(j,:)…
             + [s(j,1)*cos(heading(j+1,1)) s(j,1)*sin(heading(j+1,1))];
           right_location(j+1,:) = left_location(j+1,:)…
             - [60*cos(heading(j+1,1)) 60*sin(heading(j+1,1))];
        end

        increment(j,1) = inc(i,1);        %incremental change between servo
                                          %commands
        tweenpulse(j,1) = tween(i,1);     %pause between servo pusles
        j = j+1;

    end
end


%create matrix of information to be plotted
%the plots depict the location of the robot joints based on the robot
%commands output by the program, the speed of the robot, and the robot foot
positions

plotter2 = zeros(2*size(plotter,1),9);
%change roll every other command
%(roll repeats during the turn that pace changes)
for i=1:size(plotter,1)
    plotter2(2*i-1,1) = plotter(i,1);
    plotter2(2*i,1) = plotter(i,1);
    plotter2(2*i-1,3) = increment(i,1);         %incremental step in servo
                                                %commands changes when roll
                                                %changes
    plotter2(2*i,3) = increment(i,1);
    plotter2(2*i-1,4) = tweenpulse(i,1);        %pause after pulse changes
                                                %when roll changes
    plotter2(2*i,4) = tweenpulse(i,1);
end

%change pace every other command
%(pace repeats during the turn that roll changes)
for i=1:size(plotter,1)-1
    plotter2(2*i,2) = plotter(i,2);
    plotter2(2*i+1,2) = plotter(i,2);
    plotter2(2*i,5) = heading(i+1,1);           %heading changes when pace
                                                %changes
    plotter2(2*i+1,5) = heading(i+1,1);
    plotter2(2*i,6:7) = left_location(i+1,:);   %position of robot changes
                                                %when pace changes
    plotter2(2*i+1,6:7) = left_location(i+1,:);
    plotter2(2*i,8:9) = right_location(i+1,:);
    plotter2(2*i+1,8:9) = right_location(i+1,:);
end
plotter2(size(plotter2,1),2) = plotter(size(plotter,1),2);
                             %last pace command at end of plotting matrix
plotter2(size(plotter2,1),5) = heading(size(plotter,1),1);
                             %last heading at end of plotting matrix
plotter2(size(plotter2,1),6:7) = left_location(size(plotter,1),1:2);
                                          %last xy_location at end of
```

```
                                        %plotting matrix
plotter2(size(plotter2,1),8:9) = right_location(size(plotter,1),1:2);
plotter2(1,2) = atpace;                 %second pace command repeats atpace
plotter2(1,5:9) = [0 0 0 -60 0];        %heading and left foot remain at
zero, right foot at -60 mm
plotter3 = [atroll atpace 0 0 0 0 0 -60 0; plotter2];   %adds the initial
                                        %positions of the robot to plot
                                        %standing at attention

plotter4 = [atroll atpace 0 0 0 0 -60 0];
           %roll, pace, tweenpulse, heading, x, y

for i=2:size(plotter3,1)-2          %eliminate the last roll and pace
                                    %that had been added to vw initially
    atroll = plotter3(i-1,1);
    roll = plotter3(i,1);
    atpace = plotter3(i-1,2);
    pace = plotter3(i,2);
    increment = plotter3(i,3);
    tweenpulse = plotter3(i,4);
    atheading = plotter3(i-1,5);
    heading = plotter3(i,5);
    atleft_location = plotter3(i-1,6:7);
    atright_location = plotter3(i-1,8:9);
    left_location = plotter3(i,6:7);
    right_location = plotter3(i,8:9);
    plot_matrix = intermediate_steps(atroll, roll, atpace, pace, increment,…
      tweenpulse, atheading, heading, atleft_location, atright_location,…
      left_location, right_location);
    plotter4 = [plotter4;plot_matrix];
end


plot_robot(plotter4)
```

```
function [roll_pace, steplength, turn_angle, inc, tween] = servo_position(vw)

%This function forms a matrix of values for the roll and pace commands.
%Additionally this function creates a list of the increment and tweenpulse
%commands to move the servos at the desired speeds
```

```
%setup matrices for the function outputs
num_steps = size(vw,2);
roll_pace = zeros(num_steps,4);
steplength = zeros(num_steps,1);
turn_angle = zeros(num_steps,1);
inc = zeros(num_steps,1);
tween = zeros(num_steps,1);
atpace = 96;                        %initially standing with left foot forward
atroll = 170;                       %and no body roll/tilt


for i = 1:num_steps

  v = vw(1,i);
  w = vw(2,i);

%-----------------------------------------
%special case when v = w = 0
  if (v | w) == 0
        pause(0.1);
        T = 0.1;
  else
%-----------------------------------------
```

```
%-------------call function choose_step----------------
%find the roll and pace commands necessary to acquire s/theta that most
%closely matches v/w

    [roll1, pace1, roll2, pace2, s, theta] = choose_step (v,w,atpace);
    theta = theta*pi/180;                       %convert to radians

%create matrix of angular positions of roll servo, pace servo
    roll_pace(i,:) = [roll1 pace1 roll2 pace2];


%calculate change in PULSOUT command
    if pace1 == 0                               %if lead foot permitted
                                                %turning in desired direction
        p = abs(roll2-atroll)+abs(pace2-atpace);  %change in PULSOUT command
    else                          %if lead foot switch required before turn
        p = abs(roll1-atroll)+abs(pace1-atpace)+abs(roll2-roll1)…
            +abs(pace2-pace1);                %change in PULSOUT command
    end

%prepare atroll, atpace for future use
      atroll = roll2;                           %update atroll and atpace
      atpace = pace2;

%find the servo pulse step increments and delays
%--------------includes call to function servospeed--------------
    if (w & theta) == 0               %no turning (avoid divide by theta=0)
        pdot = v*p/s;                           %rate at which PULSOUT command
                                                %must change to produce v
        [increment, tweenpulse, pulsout_rate, error_pulsout_rate]…
            = servospeed (pdot);
        v_act = pulsout_rate*s/p;               %velocity produced
        v_error = (v_act-v)/abs(v);             %velocity error
    elseif (v & s) == 0               %turning in place (avoid divide by s=0)
        pdot = w*p/theta;                       %rate at which PULSOUT command
                                                %must change to produce w
        [increment, tweenpulse, pulsout_rate, error_pulsout_rate]…
            = servospeed (pdot);
        w_act = pulsout_rate*theta/p;           %angular velocity produced
        w_error = (w_act-w)/abs(w);             %angular velocity error
    else                                        %neither v,w,s,theta = 0
        pdot1 = v*p/s;                          %rate at which PULSOUT command
must change to produce v
        [increment1, tweenpulse1, pulsout_rate1, error_pulsout_rate1] =
servospeed (pdot1);
        v_act1 = pulsout_rate1*s/p;             %velocity produced
        v_error1 = (v_act1-v)/abs(v);           %velocity error
        w_act1 = pulsout_rate1*theta/p;         %angular velocity produced
        w_error1 = (w_act1-w)/abs(w);           %angular velocity error
        pdot2 = w*p/theta;                      %rate at which PULSOUT command
                                                %must change to produce v
        [increment2, tweenpulse2, pulsout_rate2, error_pulsout_rate2]…
            = servospeed (pdot2);
        v_act2 = pulsout_rate2*s/p;             %velocity produced
        v_error2 = (v_act2-v)/abs(v);           %velocity error
        w_act2 = pulsout_rate2*theta/p;         %angular velocity produced
        w_error2 = (w_act2-w)/abs(w);           %angular velocity error

        %select increment and tweenpulse for lowest error
        %output error message if error is too great
        if abs(v_error1)+abs(w_error1) < abs(v_error2)+abs(w_error2)
            v_error = v_error1;
            w_error = w_error1;
            increment = increment1;
```

```
            tweenpulse = tweenpulse1;
        else
            v_error = v_error2;
            w_error = w_error2;
            increment = increment2;
            tweenpulse = tweenpulse2;
        end

%set acceptable error limit
        %v_error
        limit = 1;
        if abs(v_error) > limit
            error('Desired v cannot be achieved')
        elseif abs(w_error) > limit
            error('Desired w cannot be achieved')
        end
    end
  end

  %record steplength and turn angle of each step
  steplength(i,1) = s;
  turn_angle(i,1) = theta;
  inc(i,1) = increment;
  tween(i,1) = tweenpulse;

end
```

```
function [roll1, pace1, roll2, pace2, s, theta] = choose_step (v,w,atpace)

%This function decides whether the robot is walking straight or turning.
%When turning, this function determines if the initial orientation of the
%robot legs allows turning in the desired direction.  If the orientation
%does allow the desired direction, the subroutine toroll is used.  If the
%feet need to switch orientation, the subroutine step_toroll is used.
```

```
if w > 0                            %left turn desired
    pace2 = 96;                     %move lf foot fwd (rt back) to turn left
    if atpace == 206                %right foot lead
        [roll2,s,theta] = toroll (v,w,atpace);
        roll1 = 0;                  %step to switch lead foot unneeded
        pace1 = 0;
    else                            %left lead - must step before left turn
        [roll2,s,theta] = step_toroll (v,w,atpace);
        if s > 0                    %step forward to switch lead foot
            roll1 = 138;
            pace1 = 206;
        else
            roll1 = 202;
            pace1 = 206;
        end
    end
elseif w < 0                        %right turn desired
    pace2 = 206;                    %move rt foot fwd (lf back) to turn left
    if atpace == 96                                 %left foot lead
        [roll2,s,theta] = toroll (v,w,atpace);
        roll1 = 0;
        pace1 = 0;
    else                            %right lead - must step before right turn
        [roll2,s,theta] = step_toroll (v,w,atpace);
        if s > 0
            roll1 = 202;
            pace1 = 96;
        else
```

```
            roll1 = 138;
            pace1 = 96;
        end
    end
elseif atpace == 96              %w = 0, straight walk with left foot lead
    pace2 = 206;
    [roll2,s,theta] = toroll (v,w,atpace);
    roll1 = 0;
    pace1 = 0;
else                             %w = 0, straight walk with right foot lead
    pace2 = 96;
    [roll2,s,theta] = toroll (v,w,atpace);
    roll1 = 0;
    pace1 = 0;
end
```

```
function [roll,s,theta] = toroll (v,w,atpace)

%In the case that the legs are oriented to allow turning in the desired
%direction (i.e. rt fwd/ lf turn, lf fwd/ rt turn), then this function
%calculates the roll command that will most closely match the ratio of v/w.

%create matrices
%row 1 = s/theta
%row 2 = roll command
%row 3 = s
%row 4 = theta (deg)
rt_lead = [-250 -45  33  62 150 260 450;
                 155 160 165 170 175 180 185;
            -37 -13  13  24  51  54  58;
              9  16  22  22  20  13   7];
lf_lead = [-650 -175  -50  40 170 420;
            165  170  175 180 185 190;
             54   32   14  -8 -24 -43;
             -5  -12  -16 -11  -9  -6];

%if no turning is desired, this function will produce a straight walk
if w == 0                          %avoid division by zero
    w = abs(v/(max(abs(rt_lead(1,:)))+max(abs(lf_lead(1,:)))));
                                   %guarantees v/w is large
    if atpace == 96                %expects right turn (negative w)
        w = -w;
    end
end

if atpace == 206                   %right foot lead
   if v/w < rt_lead(1,1)
      roll = 138;                  %roll command for roll left
      s = -60;                     %step back
      theta = 0;
   elseif v/w == rt_lead(1,1)
      roll = rt_lead(2,1);
      s = rt_lead(3,1);
      theta = rt_lead(4,1);
      elseif v/w > rt_lead(1,size(rt_lead,2))
      roll = 202;                  %roll command for roll right
      s = 60;                      %step forward
      theta = 0;
      else
      i=1;
      while v/w > rt_lead(1,i)
         if v/w <= rt_lead(1,i+1)
            roll = rt_lead(2,i)+(v/w-rt_lead(1,i))*(rt_lead(2,i+1)…
                  -rt_lead(2,i))/(rt_lead(1,i+1)-rt_lead(1,i));
```

```
            roll = round(roll);
            s = rt_lead(3,i)+(v/w-rt_lead(1,i))*(rt_lead(3,i+1)…
                  -rt_lead(3,i))/(rt_lead(1,i+1)-rt_lead(1,i));
            theta = rt_lead(4,i)+(v/w-rt_lead(1,i))*(rt_lead(4,i+1)…
                  -rt_lead(4,i))/(rt_lead(1,i+1)-rt_lead(1,i));
         end
          i=i+1;
      end
   end

elseif atpace == 96                        %left foot lead
   if v/w < lf_lead(1,1)
      roll = 138;                          %roll command for roll left
      s = 60;                              %step forward
      theta = 0;
   elseif v/w == lf_lead(1,1)
      roll = lf_lead(2,1);
      s = lf_lead(3,1);
      theta = lf_lead(4,1);
      elseif v/w > lf_lead(1,size(lf_lead,2))
      roll = 202;                          %roll command for roll right
      s = -60;                             %step backward
      theta = 0;
   else
      i=1;
      while v/w > lf_lead(1,i)
         if v/w <= lf_lead(1,i+1)
            roll = lf_lead(2,i)+(v/w-lf_lead(1,i))*(lf_lead(2,i+1)…
                  -lf_lead(2,i))/(lf_lead(1,i+1)-lf_lead(1,i));
            roll = round(roll);
            s = lf_lead(3,i)+(v/w-lf_lead(1,i))*(lf_lead(3,i+1)…
                  -lf_lead(3,i))/(lf_lead(1,i+1)-lf_lead(1,i));
            theta = lf_lead(4,i)+(v/w-lf_lead(1,i))*(lf_lead(4,i+1)…
                  -lf_lead(4,i))/(lf_lead(1,i+1)-lf_lead(1,i));
         end
          i=i+1;
      end
   end

else
   error('legs not fully splayed; this function will not work')

end
```

```
function [roll,s,theta] = step_toroll (v,w,atpace)

%In the case that the legs are oriented to prevent turning in the desired
%direction (i.e. rt fwd/ rt turn, lf fwd/ lf turn), then this function
%calculates the roll command that will most closely match the ratio of v/w
%after a step forward or backward has been taken.

%create matrices
%row 1 = s/theta
%row 2 = roll command
%row 3 = s
%row 4 = theta (deg)

lf_turn = [-618 -261 -122 -94 -26 -16 146 168 190 219 318 502 966;
            155  160  165 170 178 185 155 160 165 170 175 180 185;
            -97  -73  -47 -36  -7  -2  23  47  73  84 111 114 118;
              9   16   22  22  16   7   9  16  22  22  20  13   7];

rt_turn = [-1306 -439 -265 -271 -229 -162  69 134 165 354 535  984;
             165  170  175  180  185  190 165 170 175 180 185  190;
```

```
            114   92   74   52   36   17  -6 -28 -46 -68 -84 -103;
             -5  -12  -16  -11   -9   -6  -5 -12 -16 -11  -9   -6];

%separate the forward and backward stepping parts of the above matrices
%this step is necessary because the roll command cannot be interpolated
%between these two parts
for i=1:size(lf_turn,2)-1
    if lf_turn(1,i)*lf_turn(1,i+1) < 0
        bk_lf_turn = lf_turn(:,1:i);
        fd_lf_turn = lf_turn(:,i+1:size(lf_turn,2));
    end
end
for i=1:size(rt_turn,2)-1
    if rt_turn(1,i)*rt_turn(1,i+1) < 0
        fd_rt_turn = rt_turn(:,1:i);
        bk_rt_turn = rt_turn(:,i+1:size(rt_turn,2));
    end
end

%if no turning is desired, this function will produce a straight walk
if w == 0                                          %avoid division by zero
    w = abs(v/(max(abs(rt_turn(1,:)))+max(abs(lf_turn(1,:)))));
                                                   %guarantees v/w is large
end


if atpace == 96                                 %left foot lead before step
                                                %(left turn desired)
 if v/w < 0                                     %backward step desired
   if v/w < bk_lf_turn(1,1)                     %below minimum value in matrix
      roll = 138;                               %roll command for roll left
      s = -120;
      theta = 0;
   elseif v/w == bk_lf_turn(1,1)                %matches first column of matrix
      roll = bk_lf_turn(2,1);
      s = bk_lf_turn(3,1);
      theta = bk_lf_turn(4,1);
   elseif v/w > bk_lf_turn(1,size(bk_lf_turn,2))    %exceeds max value
      roll = bk_lf_turn(2,size(bk_lf_turn,2));  %roll command for nearly
                                                %turning in place
      s = bk_lf_turn(3,size(bk_lf_turn,2));
      theta = bk_lf_turn(4,size(bk_lf_turn,2));
   else                                             %interpolate values
      i=1;
      while v/w > bk_lf_turn(1,i)
         if v/w <= bk_lf_turn(1,i+1)
            roll = bk_lf_turn(2,i)+(v/w-bk_lf_turn(1,i))*(bk_lf_turn(2,i+1)…
                  -bk_lf_turn(2,i))/(bk_lf_turn(1,i+1)-bk_lf_turn(1,i));
            roll = round(roll);
            s = bk_lf_turn(3,i)+(v/w-bk_lf_turn(1,i))*(bk_lf_turn(3,i+1)…
                  -bk_lf_turn(3,i))/(bk_lf_turn(1,i+1)-bk_lf_turn(1,i));
            theta = bk_lf_turn(4,i)+(v/w-bk_lf_turn(1,i))*(bk_lf_turn(4,i+1)…
                  -bk_lf_turn(4,i))/(bk_lf_turn(1,i+1)-bk_lf_turn(1,i));
         end
         i=i+1;
      end
   end
 else                                               %forward step desired
   if v/w > fd_lf_turn(1,size(fd_lf_turn,2))        %exceeds max value
      roll = 202;                          %roll command for roll right
      s = 120;
      theta = 0;
   elseif v/w <= fd_lf_turn(1,1)           %less than or equal to minimum value
      roll = fd_lf_turn(2,1);
```

```
      s = fd_lf_turn(3,1);
      theta = fd_lf_turn(4,1);
   else                                  %interpolate values
      i=1;
      while v/w > fd_lf_turn(1,i)
         if v/w <= fd_lf_turn(1,i+1)
            roll = fd_lf_turn(2,i)+(v/w-fd_lf_turn(1,i))*(fd_lf_turn(2,i+1)…
                  -fd_lf_turn(2,i))/(fd_lf_turn(1,i+1)-fd_lf_turn(1,i));
            roll = round(roll);
            s = fd_lf_turn(3,i)+(v/w-fd_lf_turn(1,i))*(fd_lf_turn(3,i+1)…
                  -fd_lf_turn(3,i))/(fd_lf_turn(1,i+1)-fd_lf_turn(1,i));
            theta = fd_lf_turn(4,i)+(v/w-fd_lf_turn(1,i))*(fd_lf_turn(4,i+1)…
                  -fd_lf_turn(4,i))/(fd_lf_turn(1,i+1)-fd_lf_turn(1,i));
         end
         i=i+1;
      end
   end
 end

elseif atpace == 206                              %right foot lead before step
                                                  %(right turn desired)
 if v/w < 0                                       %forward step desired
   if v/w < fd_rt_turn(1,1)                       %below minimum value
      roll = 138;                                 %roll command for roll left
      s = 120;
      theta = 0;
   elseif v/w == fd_rt_turn(1,1)              %matches first column of matrix
      roll = fd_rt_turn(2,1);
      s = fd_rt_turn(3,1);
      theta = fd_rt_turn(4,1);
   elseif v/w > fd_rt_turn(1,size(fd_rt_turn,2)) %exceeds max value in matrix
      roll = fd_rt_turn(2,size(fd_rt_turn,2));   %roll command for nearly
                                                 %turning in place
      s = fd_rt_turn(3,size(fd_rt_turn,2));
      theta = fd_rt_turn(4,size(fd_rt_turn,2));
   else                                             %interpolate values
      i=1;
      while v/w > fd_rt_turn(1,i)
         if v/w <= fd_rt_turn(1,i+1)
            roll = fd_rt_turn(2,i)+(v/w-fd_rt_turn(1,i))*(fd_rt_turn(2,i+1)…
                  -fd_rt_turn(2,i))/(fd_rt_turn(1,i+1)-fd_rt_turn(1,i));
            roll = round(roll);
            s = fd_rt_turn(3,i)+(v/w-fd_rt_turn(1,i))*(fd_rt_turn(3,i+1)…
                  -fd_rt_turn(3,i))/(fd_rt_turn(1,i+1)-fd_rt_turn(1,i));
            theta = fd_rt_turn(4,i)+(v/w-fd_rt_turn(1,i))*(fd_rt_turn(4,i+1)…
                  -fd_rt_turn(4,i))/(fd_rt_turn(1,i+1)-fd_rt_turn(1,i));
         end
         i=i+1;
      end
   end
 else                                             %backward step desired
   if v/w > bk_rt_turn(1,size(bk_rt_turn,2))      %exceeds max value in matrix
      roll = 202;                                 %roll command for roll right
      s = -120;
      theta = 0;
   elseif v/w <= bk_rt_turn(1,1)  %less than or equal to minimum value
      roll = bk_rt_turn(2,1);
      s = bk_rt_turn(3,1);
      theta = bk_rt_turn(4,1);
   else                                           %interpolate values
      i=1;
      while v/w > bk_rt_turn(1,i)
         if v/w <= bk_rt_turn(1,i+1)
            roll = bk_rt_turn(2,i)+(v/w-bk_rt_turn(1,i))*(bk_rt_turn(2,i+1)…
```

```
                -bk_rt_turn(2,i))/(bk_rt_turn(1,i+1)-bk_rt_turn(1,i));
            roll = round(roll);
            s = bk_rt_turn(3,i)+(v/w-bk_rt_turn(1,i))*(bk_rt_turn(3,i+1)…
                -bk_rt_turn(3,i))/(bk_rt_turn(1,i+1)-bk_rt_turn(1,i));
            theta = bk_rt_turn(4,i)+(v/w-bk_rt_turn(1,i))*(bk_rt_turn(4,i+1)…
                -bk_rt_turn(4,i))/(bk_rt_turn(1,i+1)-bk_rt_turn(1,i));
        end
        i=i+1;
      end
   end
 end

else
   error('legs not fully splayed; this function will not work')

end
```

```
function [increment, tweenpulse, pulsout_rate, error_pulsout_rate] =
servospeed (d_pulsout_rate)

%This function receives the desired servo speed and outputs "increment" and
%"tweenpulse."
%"increment" is the variable used in the Stamp code to indicate the stepsize
%by which the PULSOUT command changes from one pulse to the next
%"tweenpulse" is the variable used to set the delay time between pulses


stepsize = [4*ones(1,4), 3*ones(1,7), 2*ones(1,12), ones(1,246)];
                                        %step size change in pulse command
pausing = [10:13, 10:16, 10:21, 10:255];        %pause between pulses (ms)
%total time of pulse = 1.6 ms + pause
pulsout_rate = 1000*(stepsize./(pausing+1.6));  %rate at which PULSOUT
                                                %command changes

%corresponds to rotational speed of servo
%d_pulsout_rate = 0; %for example

if d_pulsout_rate > pulsout_rate(1)      %if desired PULSOUT rate is too fast
    error('Desired rate of change of PULSOUT commands exceeds robot ability')
elseif d_pulsout_rate == pulsout_rate(1)
    i = 1;
elseif d_pulsout_rate < pulsout_rate(1,size(pulsout_rate,2))  %if desired
PULSOUT rate is too slow
    i = size(pulsout_rate,2);            %use slowest PULSOUT rate possible
else   %find actual PULSOUT rate produced by the Stamp that most closely
matches the desired PULSOUT rate
    i=1;
    while d_pulsout_rate < pulsout_rate(i)
        i=i+1;
    end
    if d_pulsout_rate-pulsout_rate(i) > pulsout_rate(i-1)-d_pulsout_rate
        i=i-1;
    end
end
pulsout_rate = pulsout_rate(i);
error_pulsout_rate = pulsout_rate - d_pulsout_rate;
increment = stepsize(i);
tweenpulse = pausing(i);
```

```
function [plot_matrix] = intermediate_steps(atroll, roll, atpace, pace,
increment, tweenpulse, atheading, …
      heading, atleft_location, atright_location, left_location,
right_location)
```

```
%This function creates a set of data for plotting that corresponds to each
pulse that would be generated
%by the microcontroller.
```

```
if pace-atpace == 0
    %divide each goal of roll and pace servos into incremental steps

    rolls = linspace(atroll, roll, abs(roll-atroll)/increment+1);
    paces = pace*ones(size(rolls));
    tweenpulses = tweenpulse*ones(size(rolls));
    headings = heading*ones(size(rolls));
    left_locations = [left_location(1,1)*ones(size(rolls));…
      left_location(1,2)*ones(size(rolls))];
    right_locations = [right_location(1,1)*ones(size(rolls));…
      right_location(1,2)*ones(size(rolls))];


else
    %divide each goal of roll and pace servos into incremental steps
    paces = linspace(atpace, pace, abs(pace-atpace)/increment+1);
    rolls = roll*ones(size(paces));
    tweenpulses = tweenpulse*ones(size(paces));
    headings = linspace(atheading, heading, size(paces,2));
    left_locations = [linspace(atleft_location(1,1), left_location(1,1),…
      size(paces,2)); linspace(atleft_location(1,2), left_location(1,2),…
      size(paces,2))];
    right_locations = [linspace(atright_location(1,1), right_location(1,1),…
      size(paces,2)); linspace(atright_location(1,2), right_location(1,2),…
      size(paces,2))];

end

plot_matrix = [rolls; paces; tweenpulses; headings; left_locations;
right_locations]';  %combine each list and transpose
```

```
function [] = plot_robot(plotter)

%This function simulates the robot motion.  A 3D model of the robot walks
%around a grid following the expected path.  Also, the speed of the robot
%animation reflects the speed of the actual robot.
```

```
figure
anim=plot3(0,0,0,'bo-','EraseMode','xor','MarkerSize',2);
hold on
%AXIS([XMIN XMAX YMIN YMAX ZMIN ZMAX])
axis([-300 300 -300 300 0 350])
view([37.5 30])
xlabel('x (mm)')
ylabel('y (mm)')
zlabel('z (mm)')
grid on

%servo commands for standing upright
r_stand = 170;
p_stand = 151;

%prepare matrix for 3D plotting values
robot = zeros(3*size(plotter,1),8);


for i=1:size(plotter,1)
    roll = plotter(i,1);
    pace = plotter(i,2);
    deg_roll = -1*(roll - r_stand);       %servo position (degrees)
```

```
    deg_pace = -1*(pace - p_stand);

    %data points from RSSR analysis

    x= [-32 -24   -16   -8    0     8     16    24    32];
    y= [-60 -40   -20   0     20    40    60 ];
    z= [-10.353 -8.8694 -7.2365 -5.4832 -3.6497 -1.7704 0 1.6501 3.4607;
    -9.1788 -7.712 -6.0848 -4.3487 -2.5267 -0.6589 0.88808 2.7445 4.5493;
    -7.6089 -6.1536 -4.5493 -2.8247 -1.0141 0.50993 2.3778 4.2284 6.0218;
    -6.3255 -4.8816 -3.2888 -1.5756 0 1.7418 3.5982 5.4431 7.2307;
    -5.6207 -4.1826 -2.5955 -0.88808 0.56723 2.4122 4.2685 6.1077 7.8954;
    -5.4889 -4.0565 -2.4752 -0.76776 0.68755 2.5267 4.3831 6.2223 8.0099;
    -5.8155 -4.3831 -2.796 -1.0886 0.36096 2.2059 4.068 5.9072 7.7006];

    deg_tilt = interp2(x,y,z,deg_roll,deg_pace,'linear'); %robot tilt angle
    deg_hip = deg_pace/(55/13.9);              %robot striding angle
    hip = deg_hip*pi/180;
    tilt = -deg_tilt*pi/180;
    tweenpulse = plotter(i,3);
    heading = plotter(i,4);
    left_position = plotter(i,5:6);
    right_position = plotter(i,7:8);

    %create rotation matrix for rotation about x axis
    Rx=[1   0           0;
        0   cos(tilt)   -sin(tilt);
        0   sin(tilt)   cos(tilt)];
    %create rotation matrix for rotation about z axis
    Rz=[cos(heading) -sin(heading) 0;
        sin(heading) cos(heading)  0;
        0            0             1];

%if the robot is tilting onto its right leg, set the origin on the right foot
%if the robot is tilting onto its left leg, set the origin on the left foot
if tilt > 0                                    %tilting over/onto right leg

    r_front_ankle=     [0;                    0;      0                   ];
    r_toe=             [0;                    -35;    0                   ];
    r_thin_ankle=   Rx*[-13;                  -2;     8.5                 ];
    r_wide_ankle=   Rx*[-64;                  -2;     18                  ];
    r_thin_hip=     Rx*[-13+125*sin(hip);     -2;     8.5+125*cos(hip)    ];
    r_wide_hip=     Rx*[-64+125*sin(hip);     -2;     18+125*cos(hip)     ];
    l_front_ankle=  Rx*[2*125*sin(hip);       27;     0                   ];
    l_toe=          Rx*[2*125*sin(hip);       27;     0                   ]…
      +[0; 35; 0];
    l_thin_ankle=   Rx*[-13+2*125*sin(hip);   29;     8.5                 ];
    l_wide_ankle=   Rx*[-64+2*125*sin(hip);   29;     18                  ];
    l_thin_hip=     Rx*[-13+125*sin(hip);     29;     8.5+125*cos(hip)    ];
    l_wide_hip=     Rx*[-64+125*sin(hip);     29;     18+125*cos(hip)     ];

%adjust positions of com's based on angles
    r_foot=            [-25;                  -25;    -6                  ];
    r_ankle=        Rx*[-43;                  0;      10                  ];
    r_thin_leg=     Rx*[-13+62.5*sin(hip);    -4;     8.5+62.5*cos(hip)   ];
    r_wide_leg=     Rx*[-64+77*sin(hip);      -4;     18+77*cos(hip)      ];
    l_ankle=        Rx*[-43+2*(125*sin(hip)); 27;     10                  ];
    l_thin_leg=     Rx*[-13+187.5*sin(hip);   31;     8.5+62.5*cos(hip)   ];
    l_wide_leg=     Rx*[-64+173*sin(hip);     31;     18+77*cos(hip)      ];
    torso=          Rx*[-43+125*sin(hip);     13.5;   46+125*cos(hip)     ];
    %left foot parts remain nearly parallel to ground
    %add relative distances from l_front_ankle to positions on left foot
    %l_foot= [l_front_ankle(1)-25; l_front_ankle(2)+25; l_front_ankle(3)-6];
    l_foot= [2*(125*sin(hip))-25; 27*cos(tilt)+25; 27*sin(tilt)-6];
```

```
    robot(3*i-2:3*i,1:8)=[r_toe r_front_ankle r_thin_ankle r_thin_hip…
      l_thin_hip l_thin_ankle l_front_ankle l_toe];
    %adjust x positions relative to center
    robot(3*i-1,:) = robot(3*i-1,:) - 13.5;
    robot(3*i-2:3*i,:) = Rz*robot(3*i-2:3*i,:);
    robot(3*i-2,:) = robot(3*i-2,:) + right_position(1,1);
    robot(3*i-1,:) = robot(3*i-1,:) + right_position(1,2);

    else
    %tilting over/onto left leg

    l_front_ankle=     [0;                    0;     0                    ];
    l_toe=               [0;                           35;    0
           ];
    l_thin_ankle=   Rx*[-13;                  2;     8.5                  ];
    l_wide_ankle=   Rx*[-64;                  2;     18                   ];
    l_thin_hip=     Rx*[-13-125*sin(hip);     2;     8.5+125*cos(hip)     ];
    l_wide_hip=     Rx*[-64-125*sin(hip);     2;     18+125*cos(hip)      ];
    r_front_ankle=  Rx*[-2*125*sin(hip);      -27;   0                    ];
    r_toe=          Rx*[-2*125*sin(hip);      -27;   0                    ]…
      +[0; -35; 0];
    r_thin_ankle=   Rx*[-13-2*125*sin(hip);   -29;   8.5                  ];
    r_wide_ankle=   Rx*[-64-2*125*sin(hip);   -29;   18                   ];
    r_thin_hip=     Rx*[-13-125*sin(hip);     -29;   8.5+125*cos(hip)     ];
    r_wide_hip=     Rx*[-64-125*sin(hip);     -29;   18+125*cos(hip)      ];

    %adjust positions of com's based on angles
    l_foot=           [-25;                  25;    -6                  ];
    l_ankle=        Rx*[-43;                  0;     10                   ];
    l_thin_leg=     Rx*[-13-62.5*sin(hip);    4;     8.5+62.5*cos(hip)    ];
    l_wide_leg=     Rx*[-64-77*sin(hip);      4;     18+77*cos(hip)       ];
    r_ankle=        Rx*[-43-2*(125*sin(hip)); -27;   10                   ];
    r_thin_leg=     Rx*[-13-187.5*sin(hip);   -31;   8.5+62.5*cos(hip)    ];
    r_wide_leg=     Rx*[-64-173*sin(hip);     -31;   18+77*cos(hip)       ];
    torso=          Rx*[-43-125*sin(hip);     -13.5; 46+125*cos(hip)      ];
    %right foot parts remain nearly parallel to ground
    %add relative distances from l_front_ankle to positions on right foot
    %r_foot= [r_front_ankle(1)-25;  r_front_ankle(2)-25; r_front_ankle(3)-6];
    r_foot= [-2*(125*sin(hip))-25; -27*cos(tilt)-25; -27*sin(tilt)-6];

    robot(3*i-2:3*i,1:8)=[r_toe r_front_ankle r_thin_ankle r_thin_hip…
      l_thin_hip… l_thin_ankle l_front_ankle l_toe];
    robot(3*i-1,:) = robot(3*i-1,:) + 13.5;
    robot(3*i-2:3*i,:) = Rz*robot(3*i-2:3*i,:);
    robot(3*i-2,:) = robot(3*i-2,:) + left_position(1,1);
    robot(3*i-1,:) = robot(3*i-1,:) + left_position(1,2);

    end
end

robot=robot';
for i=1:size(plotter,1)
    set(anim,'XData',robot(:,3*i-2),'YData',robot(:,3*i-1),…
      'ZData',robot(:,3*i))
    drawnow
    pause(tweenpulse/1000)
end
```

Appendix B: Stamp I microcontroller program

```
'walk five steps forward
'
'Define the constants and pin allocations
SYMBOL servoroll        =6      'Roll servo connected to pin 6
SYMBOL servopace        =7      'Pace servo connected to pin 7
SYMBOL atroll           =b1
SYMBOL roll             =b2
SYMBOL atpace                   =b3
SYMBOL pace             =b4
SYMBOL atX              =b5
SYMBOL toX              =b6
SYMBOL servoX                   =b7
SYMBOL i                =b9     'loop counter in init
SYMBOL increment        =b10    'step change in PULSOUT commands
SYMBOL tweenpulse       =b11    'pause after servo pulse
'=====================================================================

init:   atroll =170             'Stand to attention for approx 2 seconds
        atpace =96
        FOR i=0 to 50
          PULSOUT servoroll,atroll
          PULSOUT servopace,atpace
          PAUSE 10
        NEXT
start:
        toX =138                        'roll command
        increment =1
        tweenpulse =20
        GOSUB doroll
        toX =202                        'roll command
        tweenpulse =16
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =202                        'roll command
        GOSUB doroll
        toX =138                        'roll command

        GOSUB doroll

doroll:
        atX =atroll
        servoX =servoroll
        GOSUB domove
        atroll =atX
        atX = atpace
        toX = 302-atpace
        servoX =servopace
        GOSUB domove
        atpace =atx
        RETURN
```

```
domove:                 IF atX=toX THEN endmove
                IF atX<toX THEN addspeed  'else subtract speed
                atX =atX-increment
                IF atX<toX THEN gobacka
                GOTO Spulse
addspeed:       atX =atX+increment
                IF atX>toX THEN gobackb
Spulse:         PULSOUT servoX,atX
                PAUSE tweenpulse
                GOTO domove
gobacka:        atX =atX+increment
                GOTO endmove
gobackb:        atX =atX-increment
endmove:        RETURN
'----------------------- end program -------------------
```

```
'walk forward in counterclockwise arc
'
'Define the constants and pin allocations
SYMBOL servoroll        =6      'Roll servo connected to pin 6
SYMBOL servopace        =7      'Pace servo connected to pin 7
SYMBOL atroll           =b1
SYMBOL roll             =b2
SYMBOL atpace                   =b3
SYMBOL pace             =b4
SYMBOL atX              =b5
SYMBOL toX              =b6
SYMBOL servoX                   =b7
SYMBOL i                =b9     'loop counter in init
SYMBOL increment        =b10    'step change in PULSOUT commands
SYMBOL tweenpulse       =b11    'pause after servo pulse
'=====================================================================


init:   atroll =170             'Stand to attention for approx 2 seconds
        atpace =96
        FOR i=0 to 50
          PULSOUT servoroll,atroll
          PULSOUT servopace,atpace
          PAUSE 10
        NEXT
start:
        toX =138                        'roll command
        increment =1
        tweenpulse =26
        GOSUB doroll
        toX =167                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =167                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =167                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =167                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =167                        'roll command
        GOSUB doroll


doroll:
        atX =atroll
        servoX =servoroll
        GOSUB domove
        atroll =atX
        atX = atpace
```

```
        toX = 302-atpace
        servoX =servopace
        GOSUB domove
        atpace =atx
        RETURN

domove:             IF atX=toX THEN endmove
                IF atX<toX THEN addspeed  'else subtract speed
                atX =atX-increment
                IF atX<toX THEN gobacka
                GOTO Spulse
addspeed:       atX =atX+increment
                IF atX>toX THEN gobackb
Spulse:         PULSOUT servoX,atX
                PAUSE tweenpulse
                GOTO domove
gobacka:        atX =atX+increment
                GOTO endmove
gobackb:        atX =atX-increment
endmove:        RETURN
'----------------------- end program --------------------
```

```
'walk backward in clockwise arc
'
'Define the constants and pin allocations
SYMBOL servoroll        =6      'Roll servo connected to pin 6
SYMBOL servopace        =7      'Pace servo connected to pin 7
SYMBOL atroll           =b1
SYMBOL roll             =b2
SYMBOL atpace                   =b3
SYMBOL pace             =b4
SYMBOL atX              =b5
SYMBOL toX              =b6
SYMBOL servoX                   =b7
SYMBOL i                =b9     'loop counter in init
SYMBOL increment        =b10    'step change in PULSOUT commands
SYMBOL tweenpulse       =b11    'pause after servo pulse
'====================================================================


init:   atroll =170             'Stand to attention for approx 2 seconds
        atpace =96
        FOR i=0 to 50
          PULSOUT servoroll,atroll
          PULSOUT servopace,atpace
          PAUSE 10
        NEXT
start:
        toX =186                        'roll command
        increment =1
        tweenpulse =22
        GOSUB doroll
        toX =138                        'roll command
        tweenpulse =15
        GOSUB doroll
        toX =176                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        tweenpulse =16
        GOSUB doroll
        toX =176                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =176                        'roll command
        GOSUB doroll
        toX =138                        'roll command
        GOSUB doroll
        toX =176                        'roll command
        GOSUB doroll

doroll:
        atX =atroll
        servoX =servoroll
        GOSUB domove
        atroll =atX
        atX = atpace
        toX = 302-atpace
```

```
        servoX =servopace
        GOSUB domove
        atpace =atx
        RETURN

domove:                 IF atX=toX THEN endmove
                IF atX<toX THEN addspeed  'else subtract speed
                atX =atX-increment
                IF atX<toX THEN gobacka
                GOTO Spulse
addspeed:       atX =atX+increment
                IF atX>toX THEN gobackb
Spulse:         PULSOUT servoX,atX
                PAUSE tweenpulse
                GOTO domove
gobacka:        atX =atX+increment
                GOTO endmove
gobackb:        atX =atX-increment
endmove:        RETURN
'----------------------- end program -------------------
```

Appendix C: Lego biped construction

Two-legged walker constructed with Lego Mindstorms Invention System 2.0

Joseph Trout

Virginia Tech
Fall, 2001

Introduction:

The purpose for building this robot was to explore a few topics involved in the study of biomimetic robots. Biomimetic robots have elements in their design that imitate characteristics of living creatures, such as their ability to walk. Walking is the mode of locomotion used by many land animals, and two-legged walking in particular mimics human locomotion. Although the movement of the robot constructed for this project is not nearly as complex as human walking, the concepts used in the design provided a way to become familiar with the considerations that would be important in designing more sophisticated walking robots.

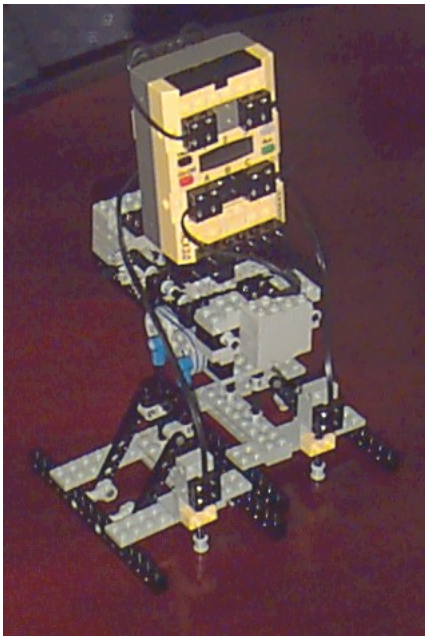


Image of the two-legged walker

Function:

The robot is designed to shift its weight over the left and right feet alternately. The Lego RCX controller was manipulated to accomplish this desired shifting of the weight. Because the

RCX contains the batteries for the robot, the RCX weighs more than any other element in the Lego robotics kit. One motor was used to tilt the RCX back-and-forth alternately over the left and right legs. A second motor was used to continuously drive the legs in a circular motion. The legs were linked to the same drive shafts. However, the legs were aligned out of phase so that one leg would move forward and up while the other leg would move backward and down. By shifting the RCX over one leg while that leg pushed down, the other leg could lift up and step forward. After stepping forward with the raised leg, the RCX would then be tilted over the raised leg, thus, moving the position of the robot forward and allowing the other leg to lift up and step forward. In this way, the robot could walk forward in a waddling motion as its weight shifted from side-to-side.

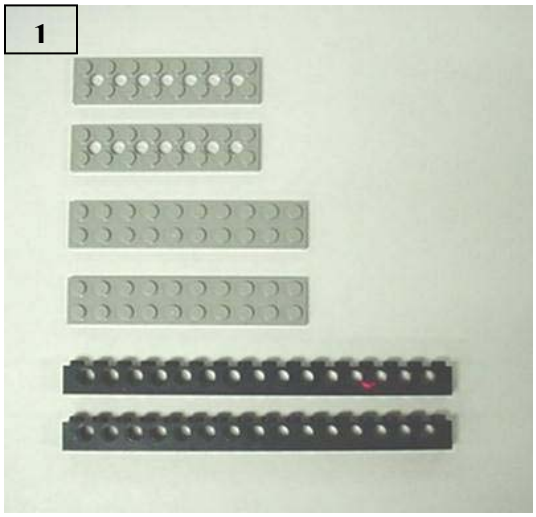
The timing issue has not been solved perfectly. Occasionally, the RCX can get out of sink with the legs. Touch sensors were attached to the feet to provide an indication of when each foot was in contact with the ground. Using the programming interface provided with the Lego system, a program was written to control the motion of the robot. First the motor on the hip section of the robot was set to run continuously, causing the legs to move in a constant circular motion. Then the program checked which foot was touching the ground by monitoring the touch sensors. After recognizing which foot was on the ground, the program waited for approximately one second to allow the other foot to move forward. After the delay had expired, the controller commanded the motor on the waist to tilt the RCX over the foot that stepped forward. When the program recognized that the touch sensor on the other foot had made contact with the ground, the program waited again for the first foot to step forward before tilting the RCX in the opposite direction. The program was set to run through an infinite loop, repeating the alternating tilting and stepping pattern.

Construction:

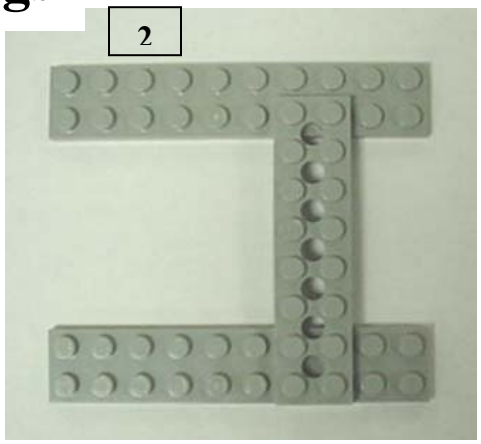
The following images provide step-by-step directions for constructing the walking robot. Beginning with the construction of the legs, the instructions then describe the hip and waist, the attachments to the RCX controller, and the final assembly of the robot.

Legs

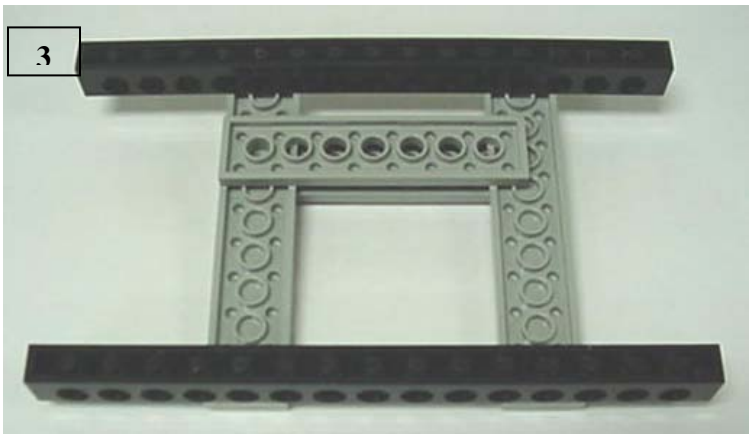
1



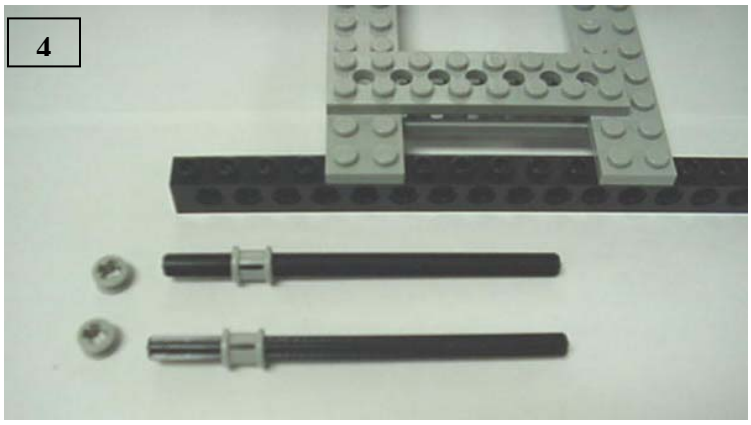
2

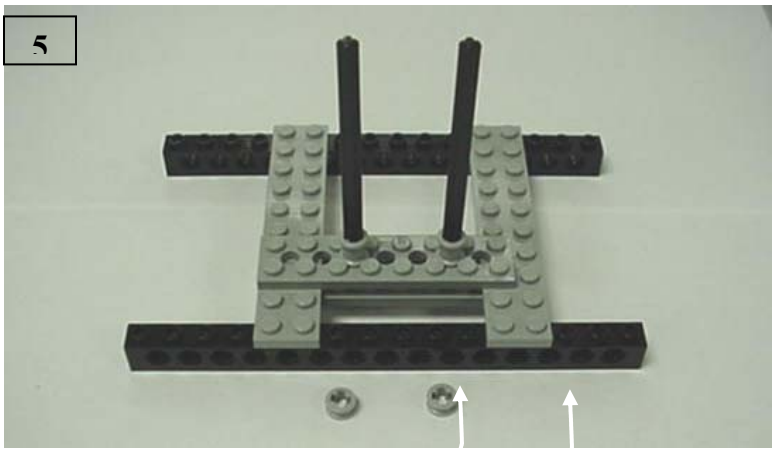


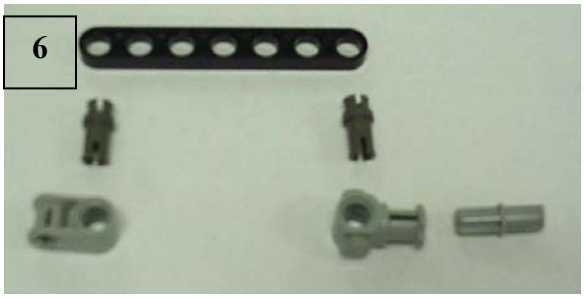
3



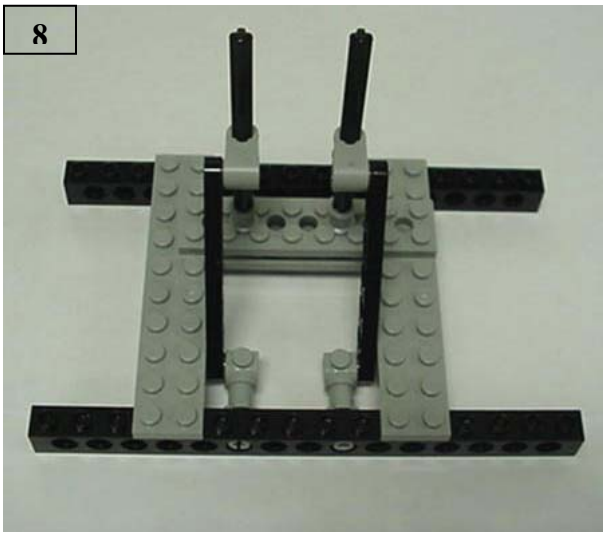
4

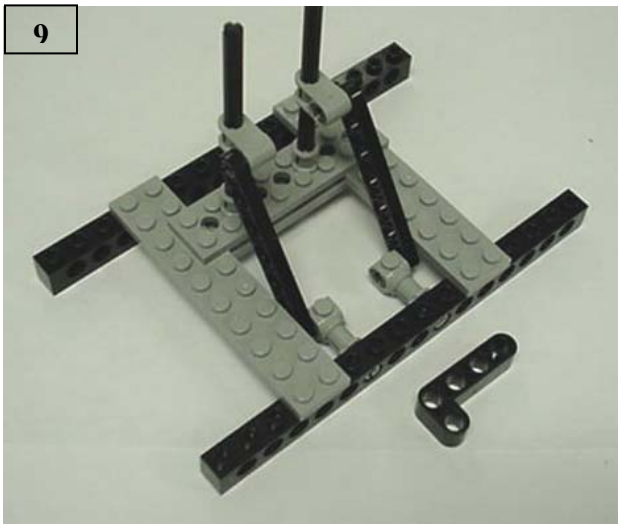



5

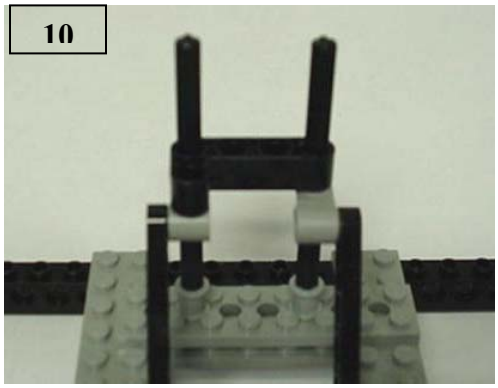

6


7


8

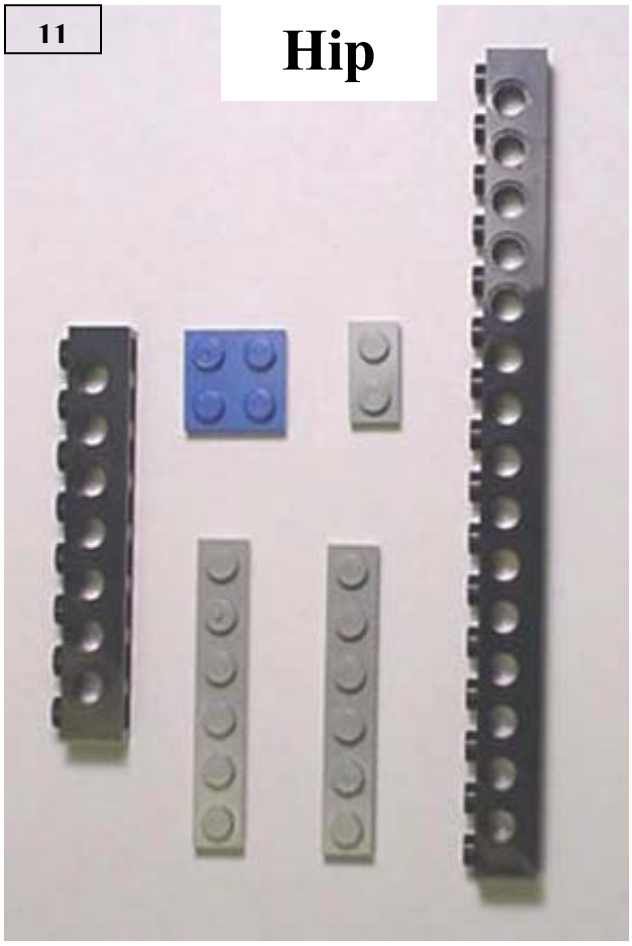

9

10



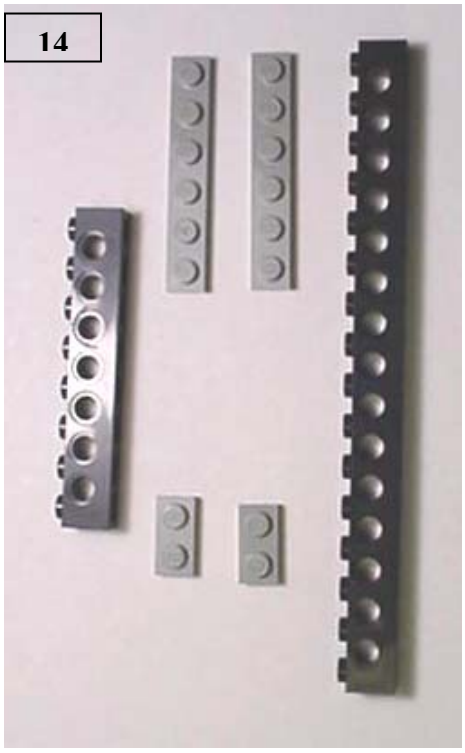
11

Hip

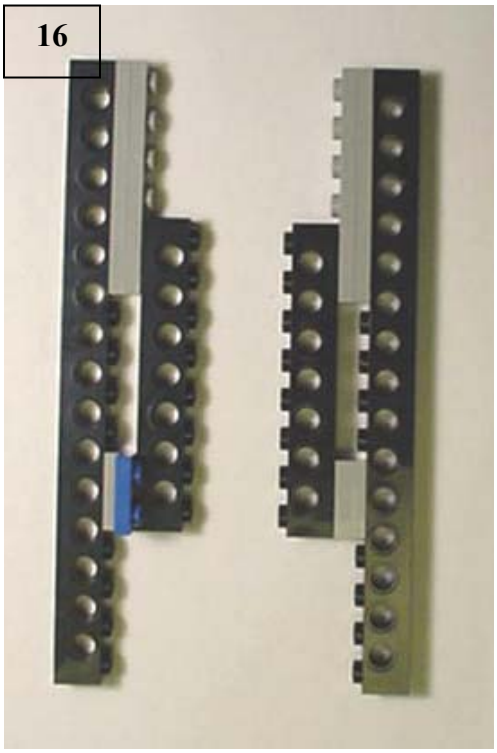


12

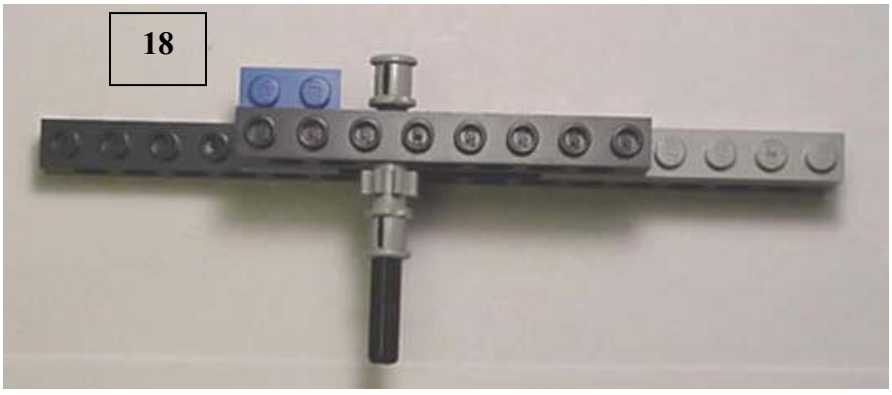
13

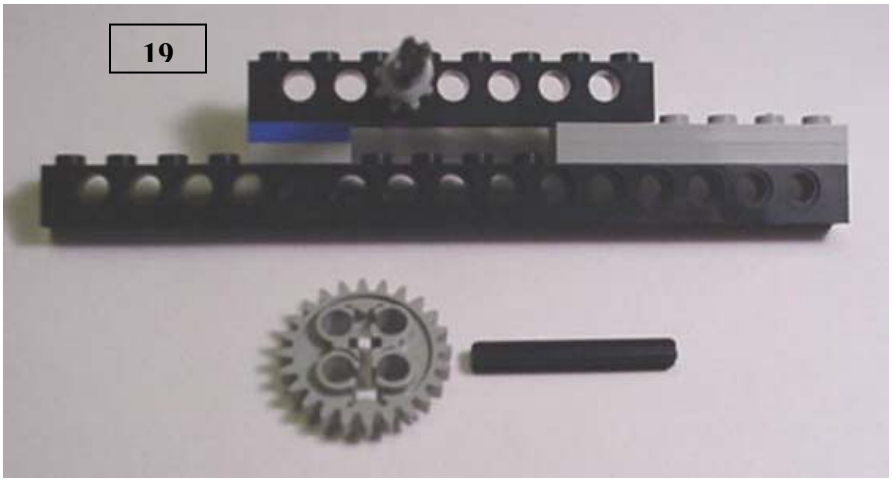

14

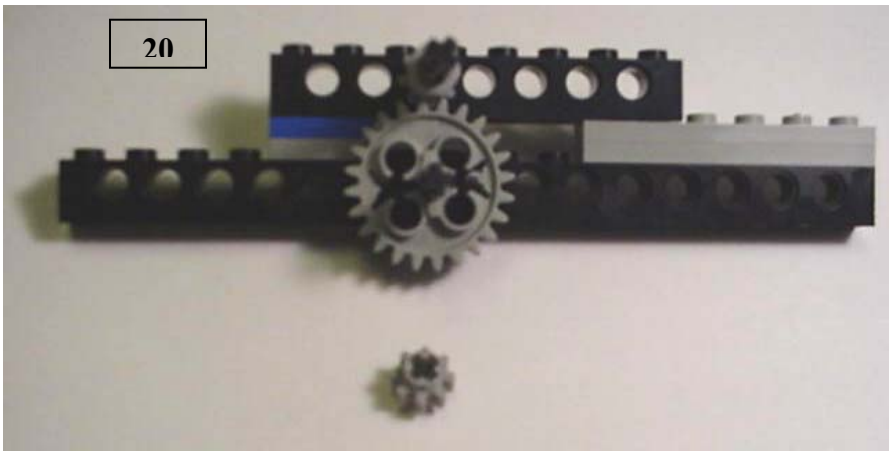

15

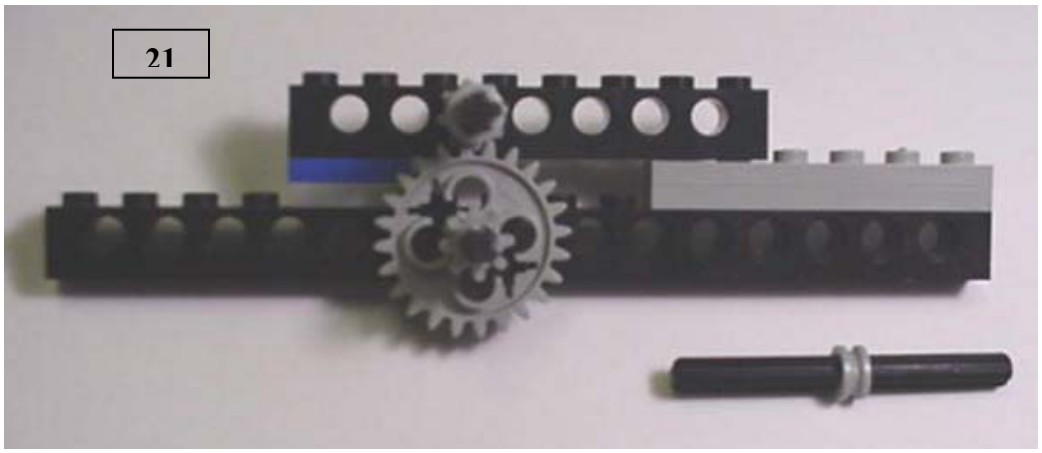

16

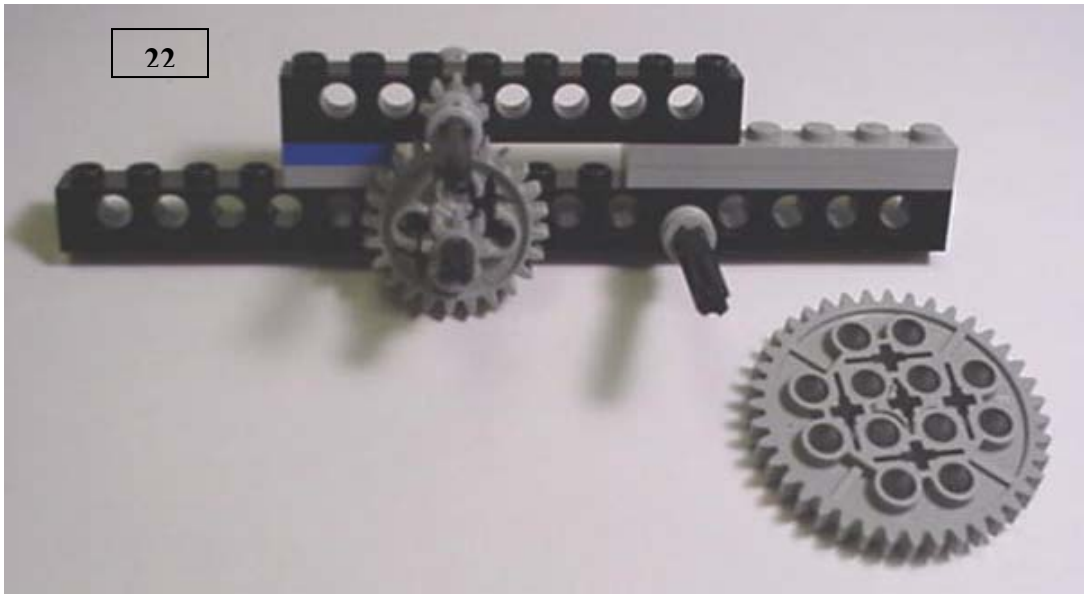

17


18

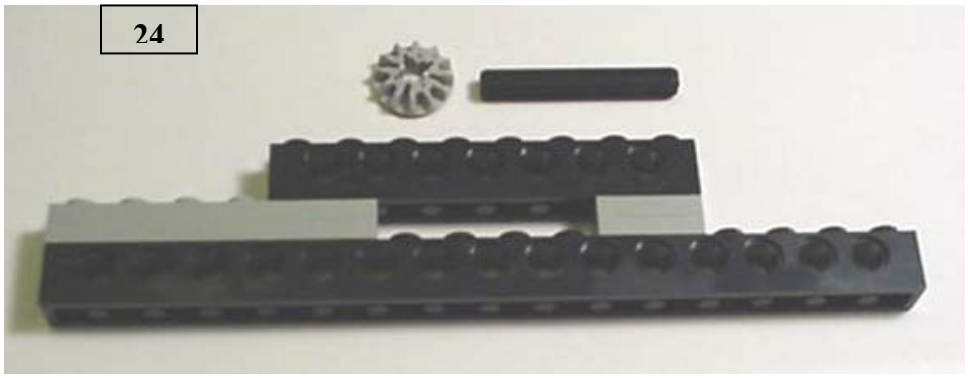

19

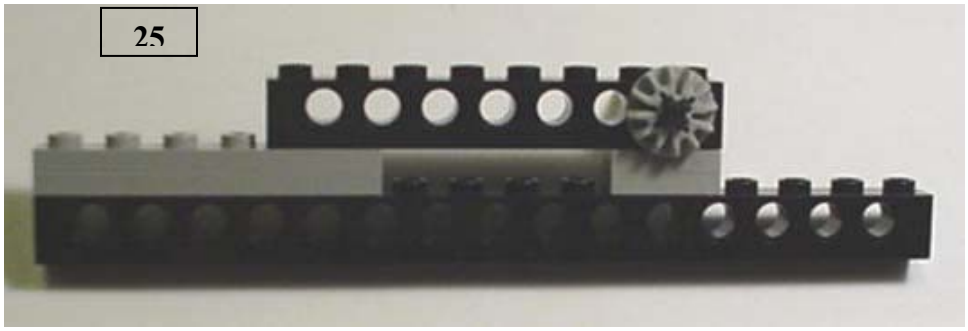

20

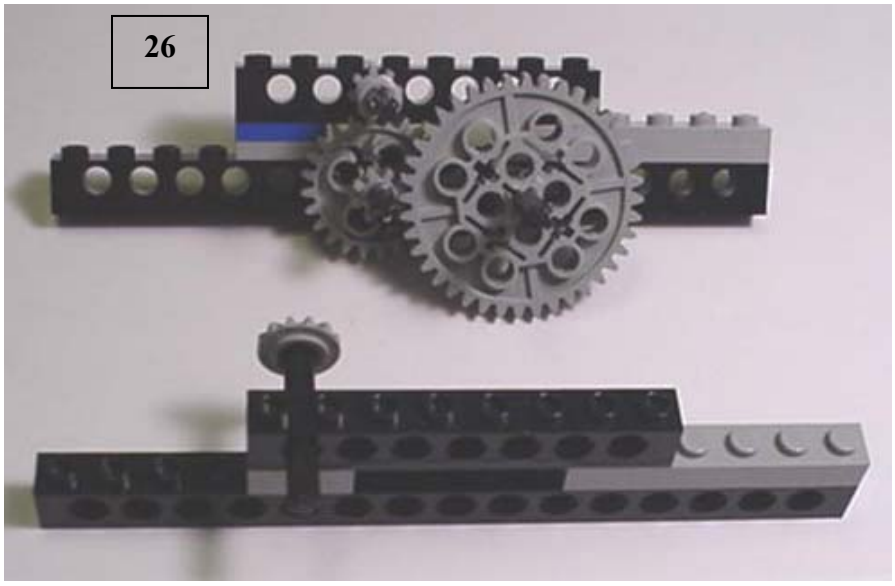

21

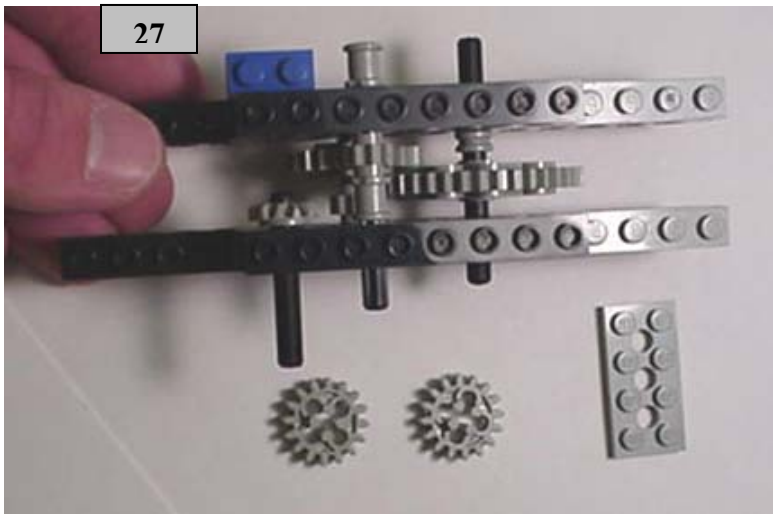

22

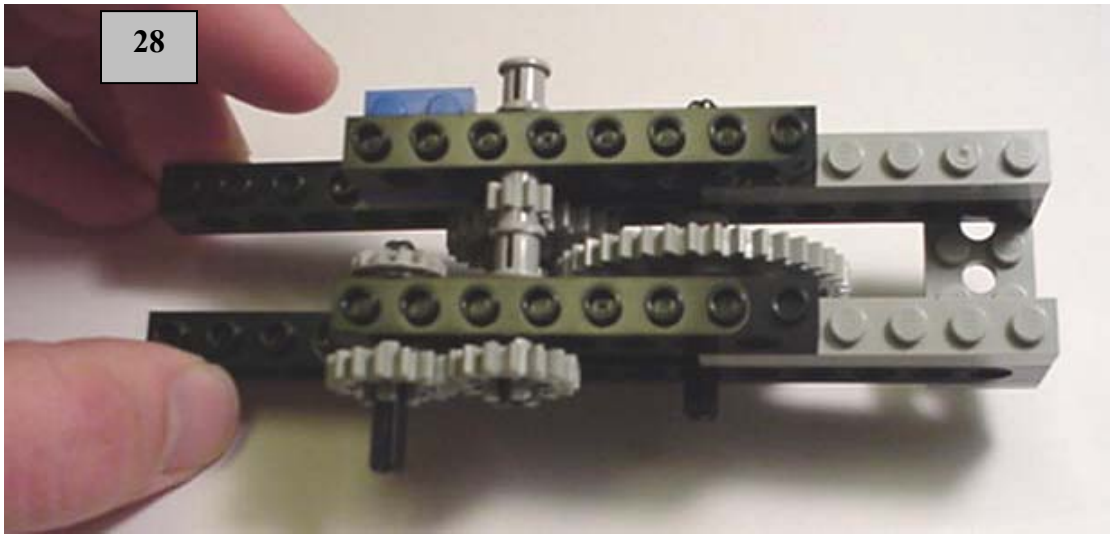

23

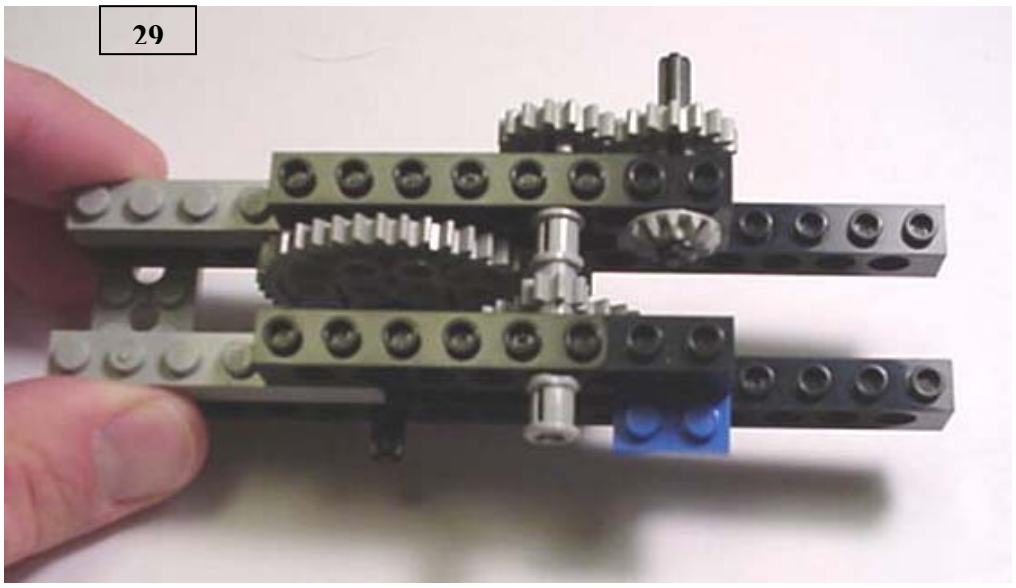

24

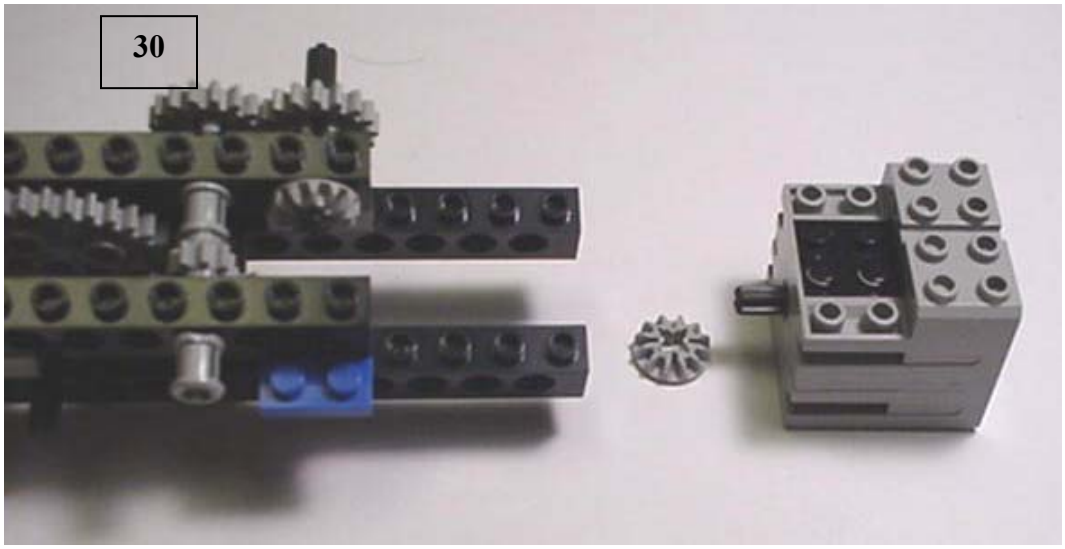

25

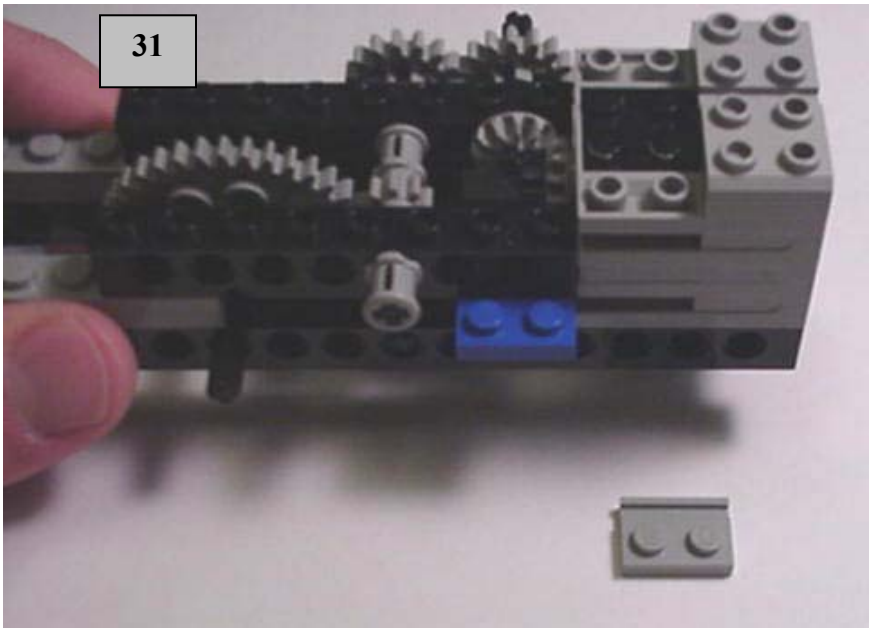

26

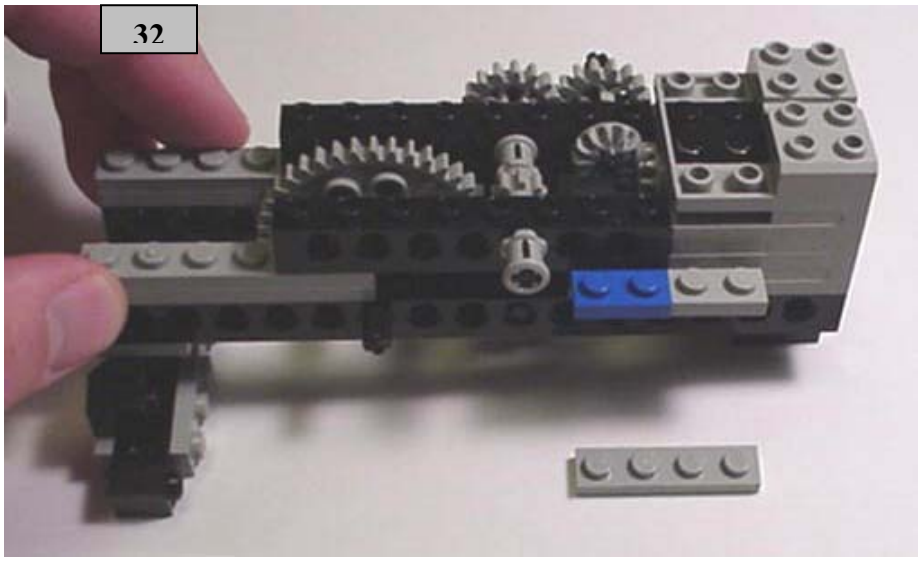

27

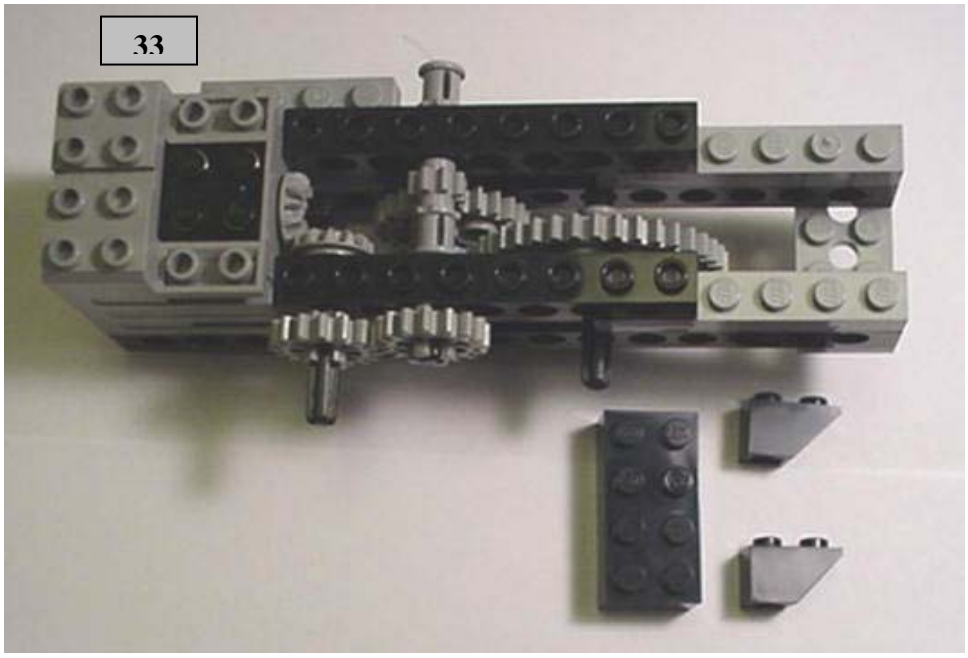

28

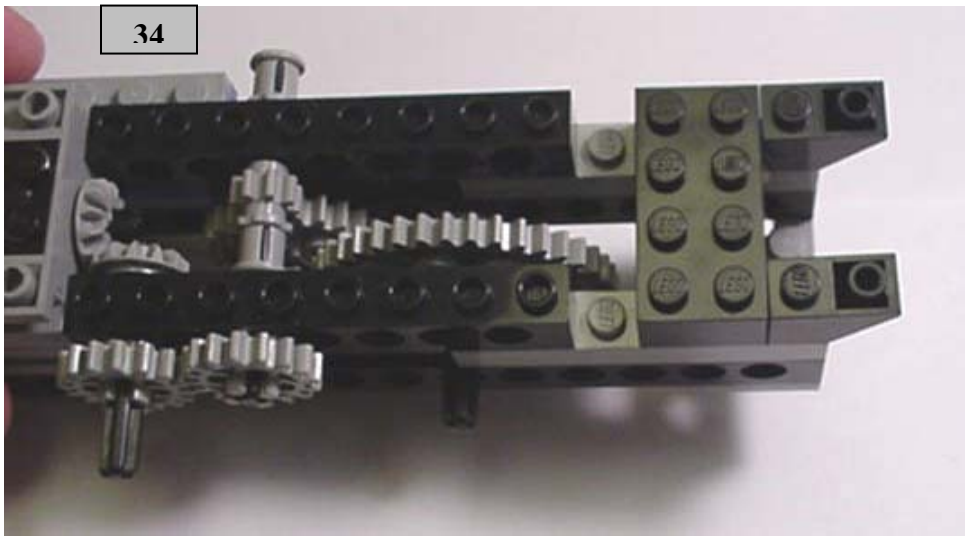

29

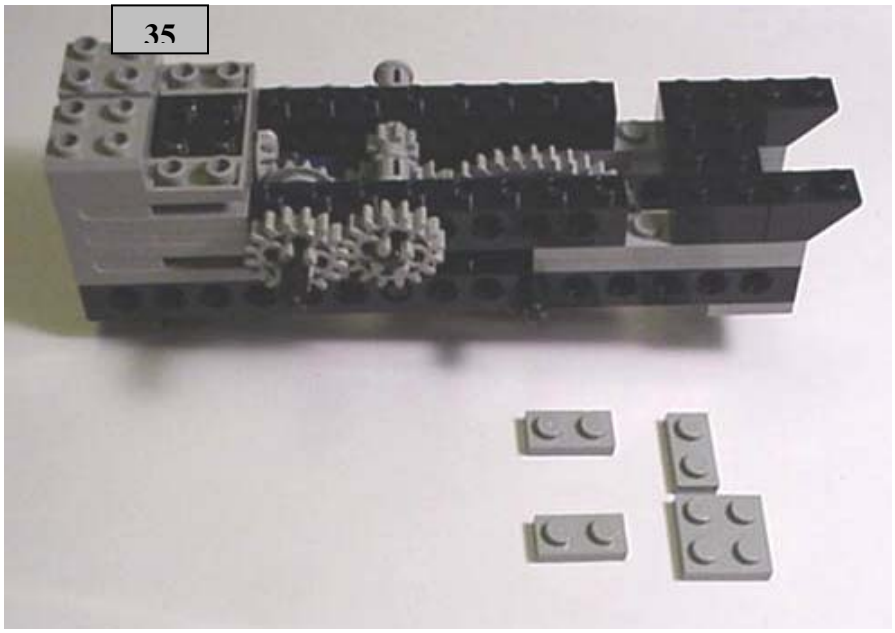

30

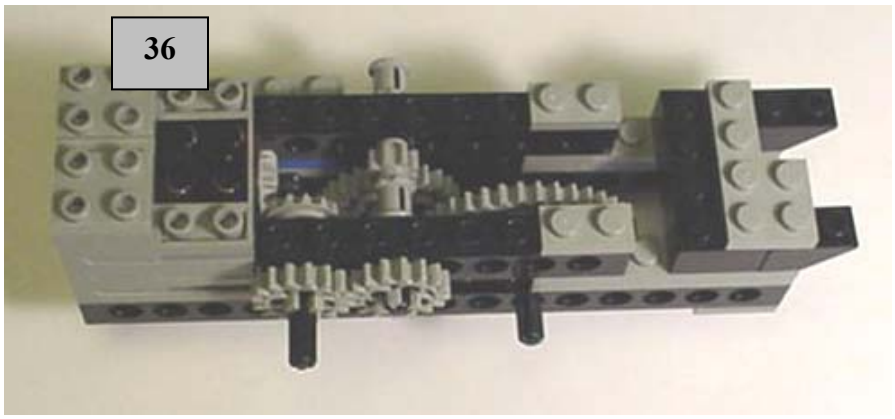

31

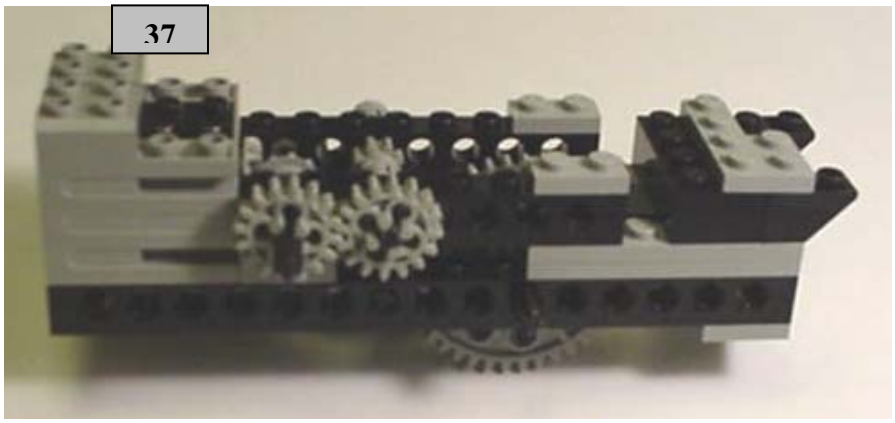

32

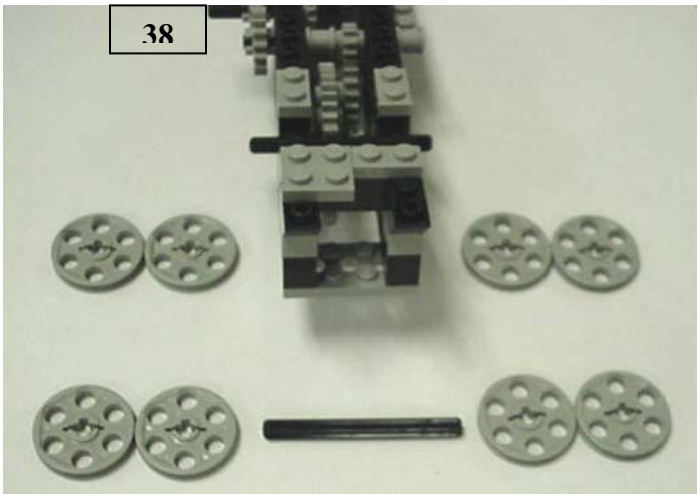

33

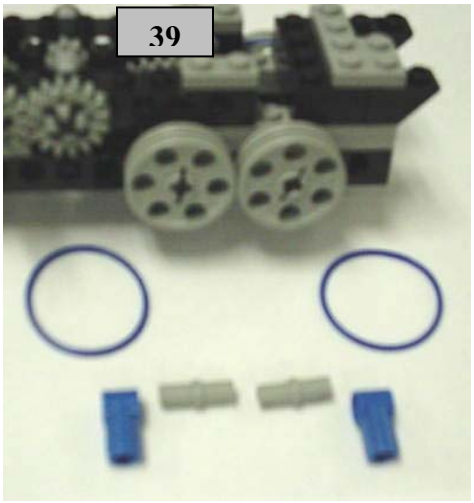

34

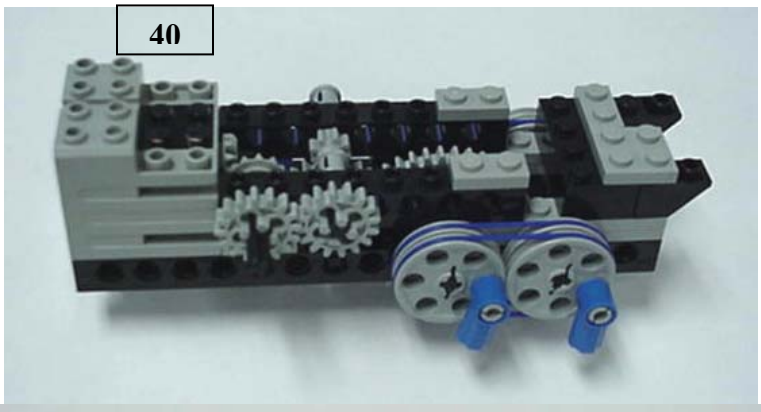

35

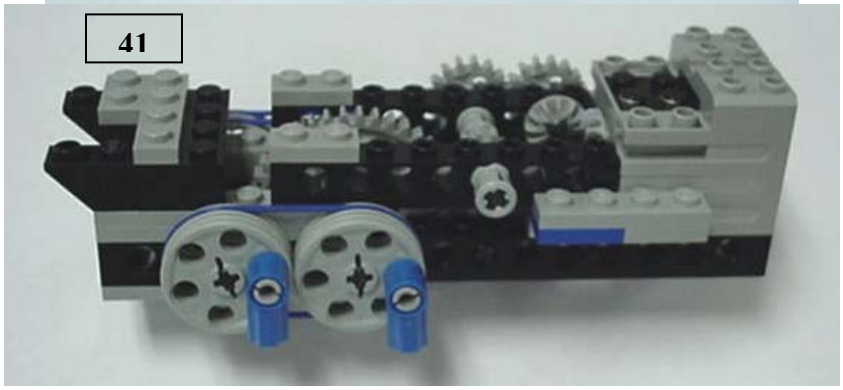

36


37


38


39


40

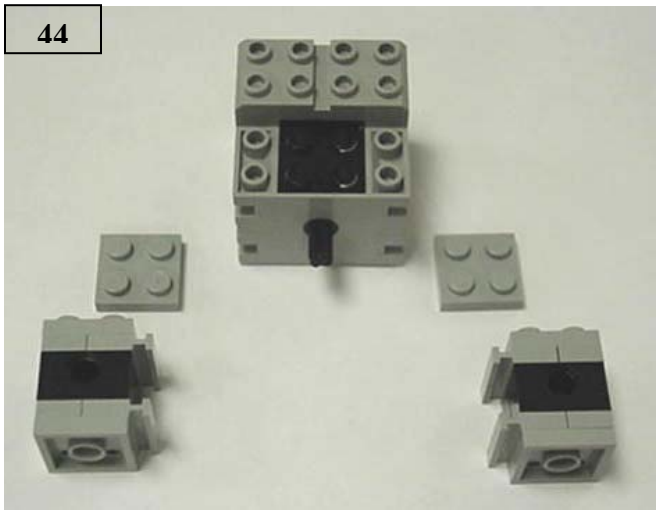

41

Waist

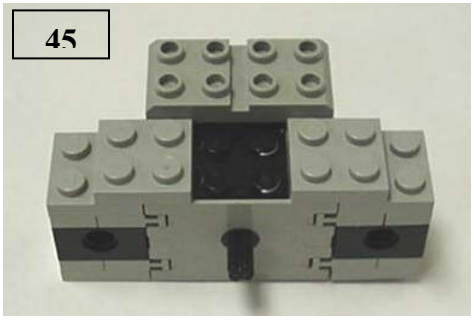
42

43

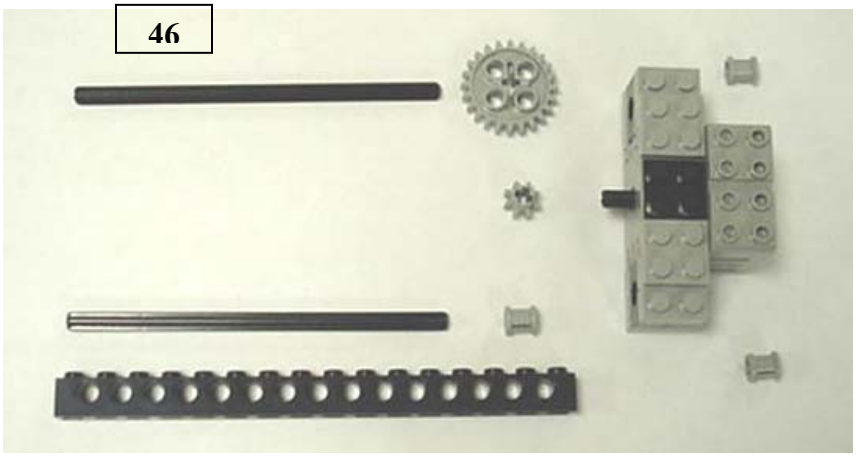
44



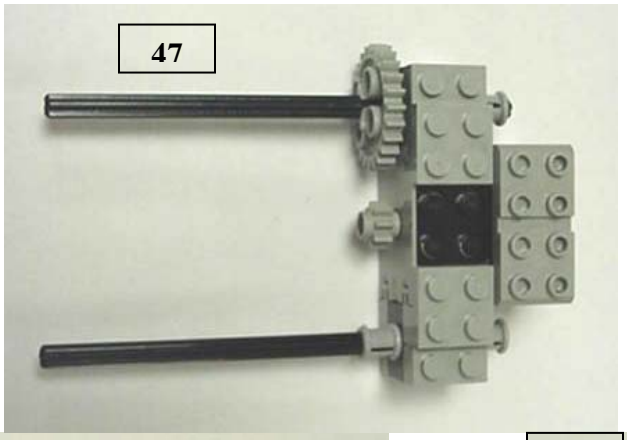
45



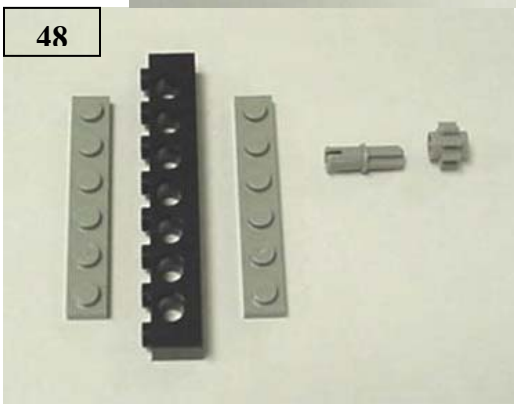
46



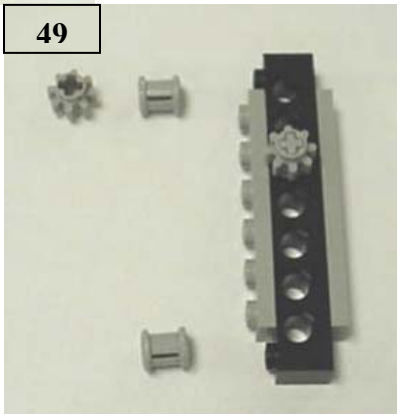
47



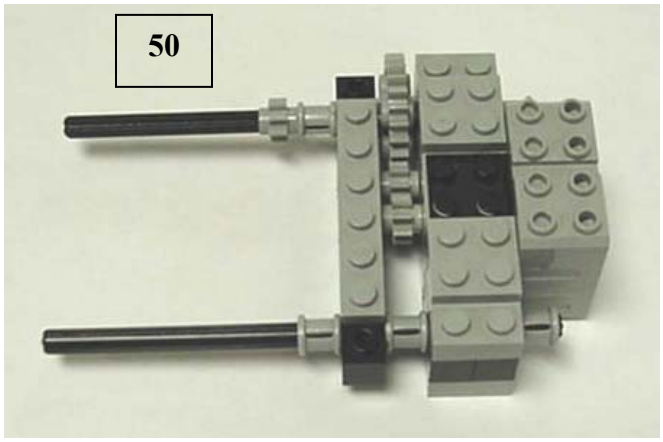
48



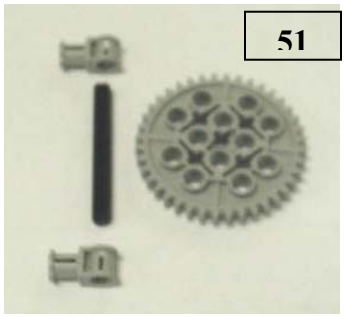
49



50

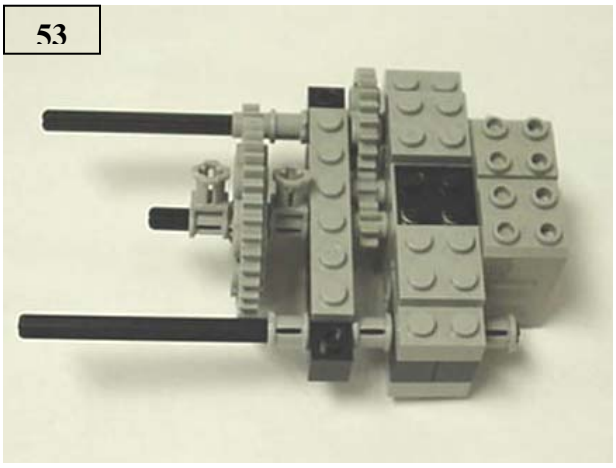


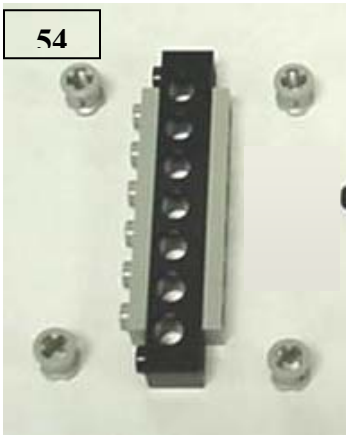
51

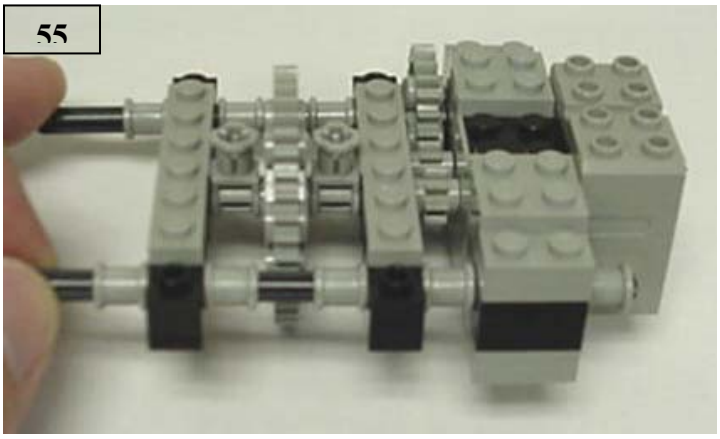


52

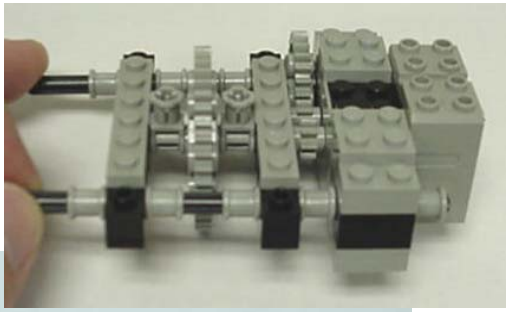
53

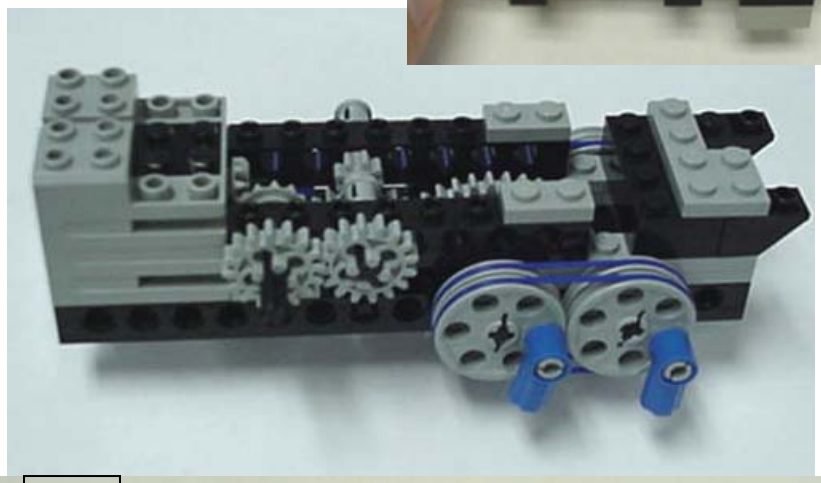


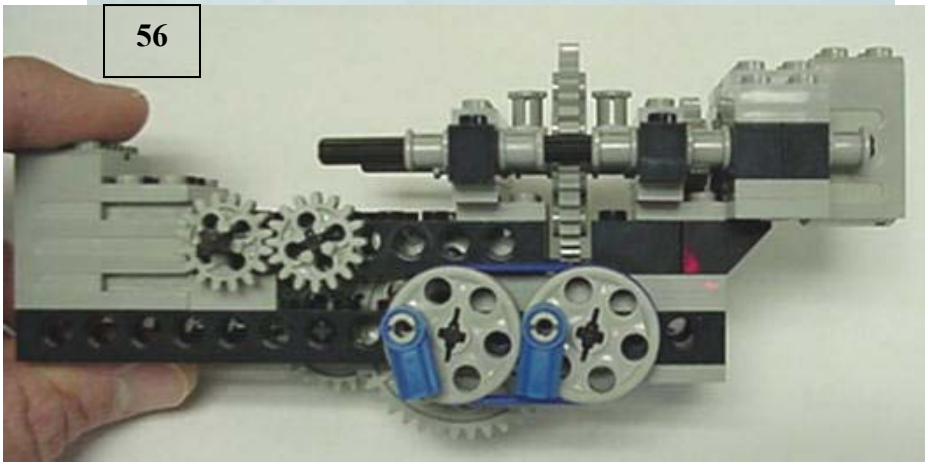




Hip and Waist Assembly



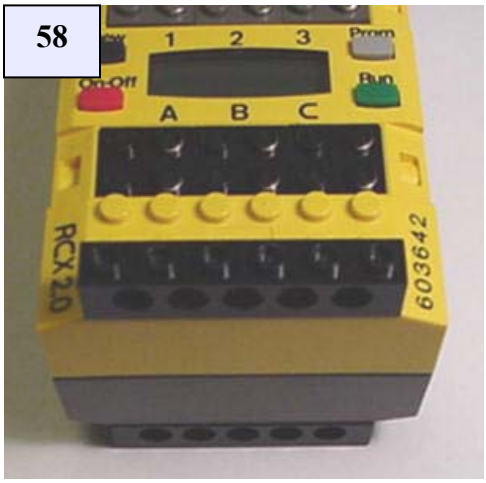




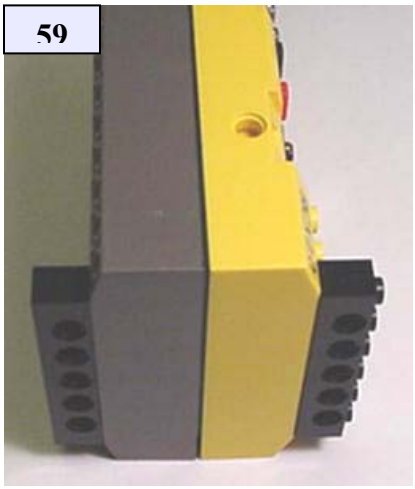
RCX Attachment

57

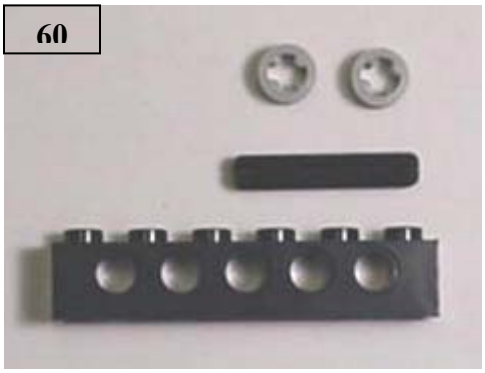
58



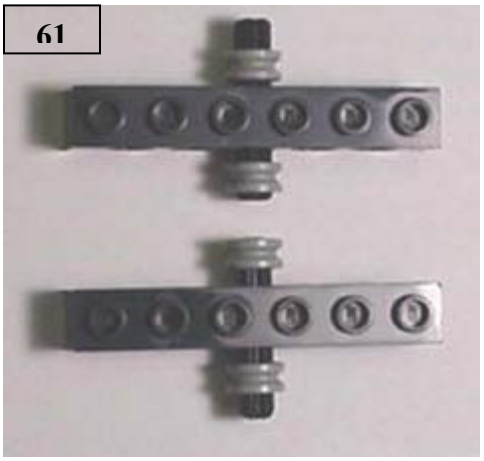
59



60

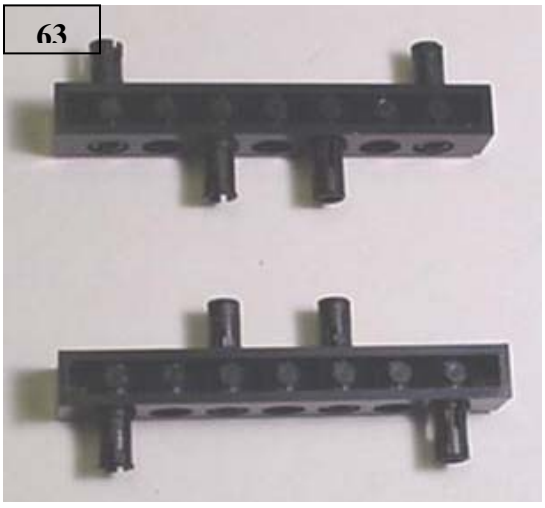


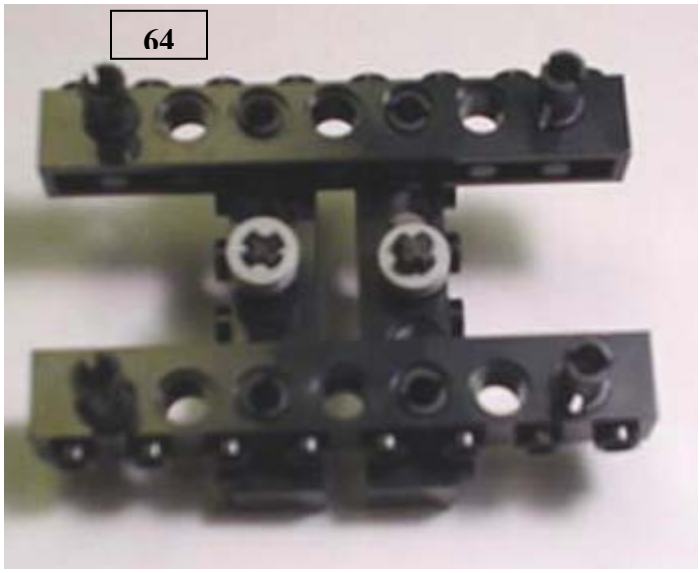
61

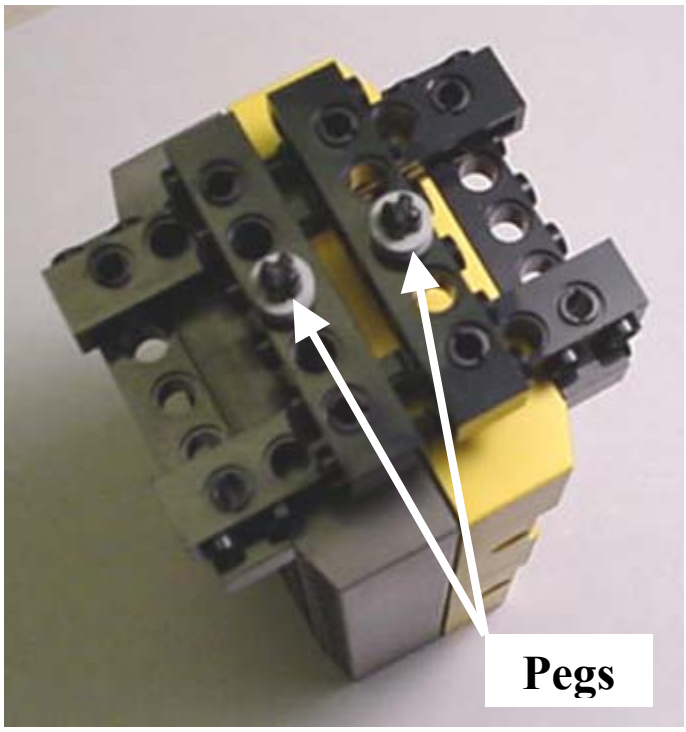


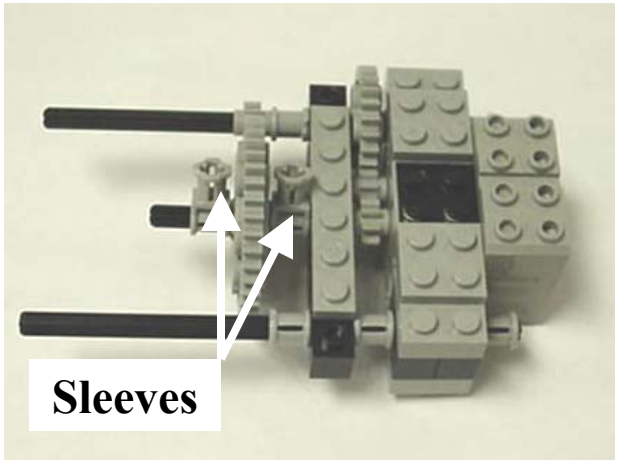
62

63



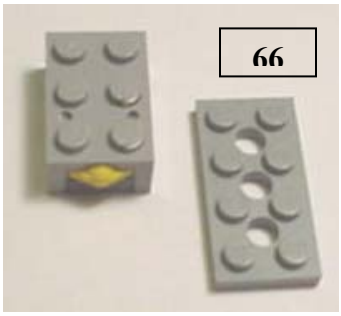


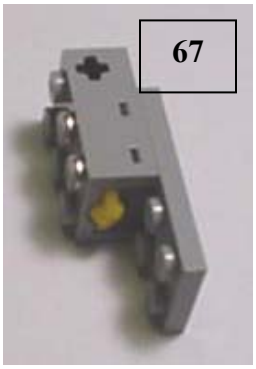


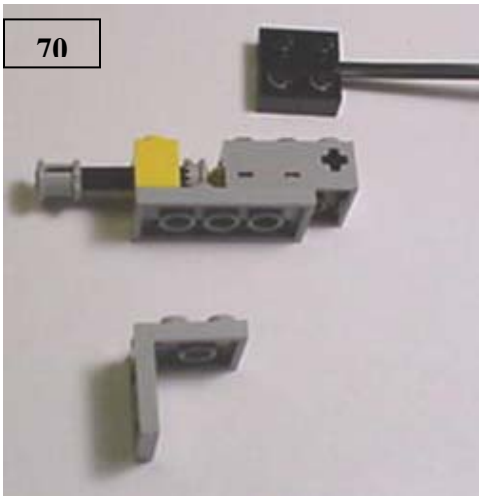


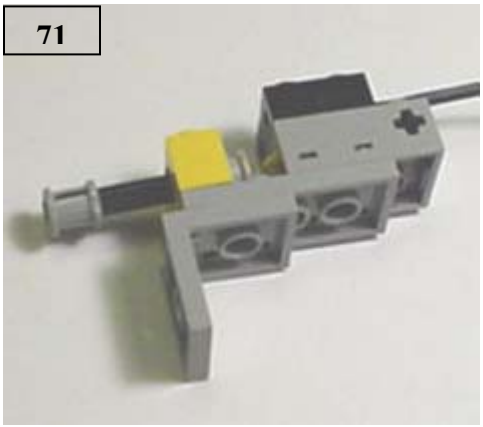
65. Attach RCX to waist by inserting pegs into sleeves.

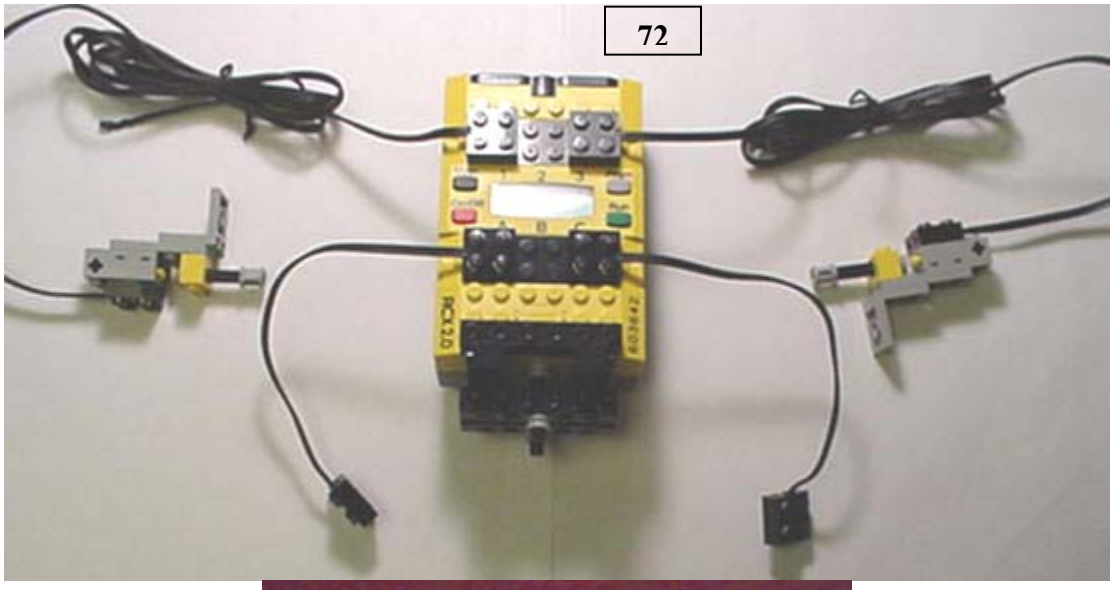
Touch Sensors

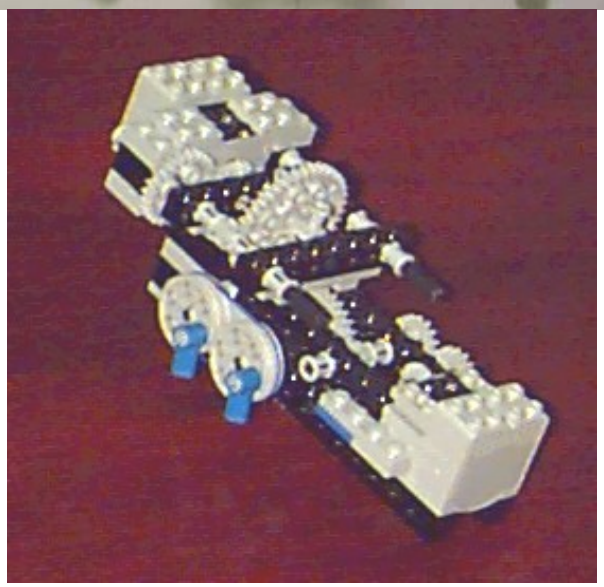




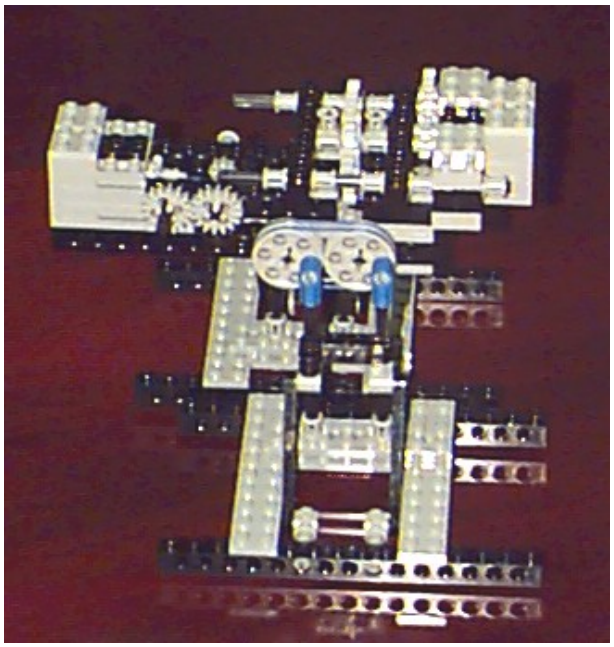




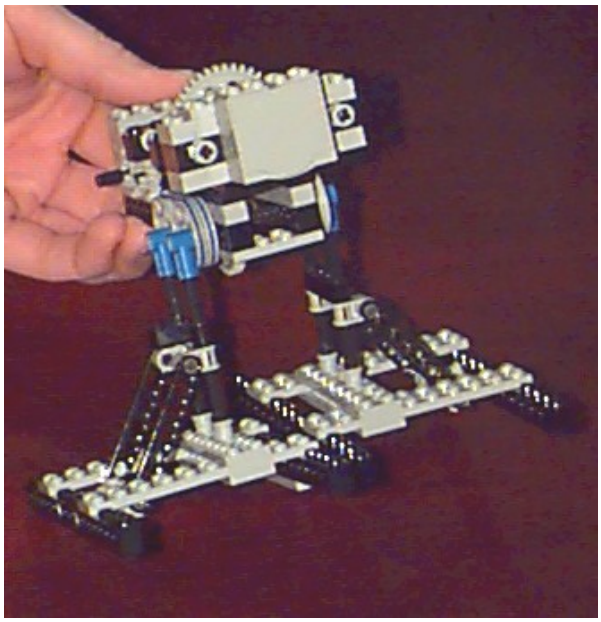




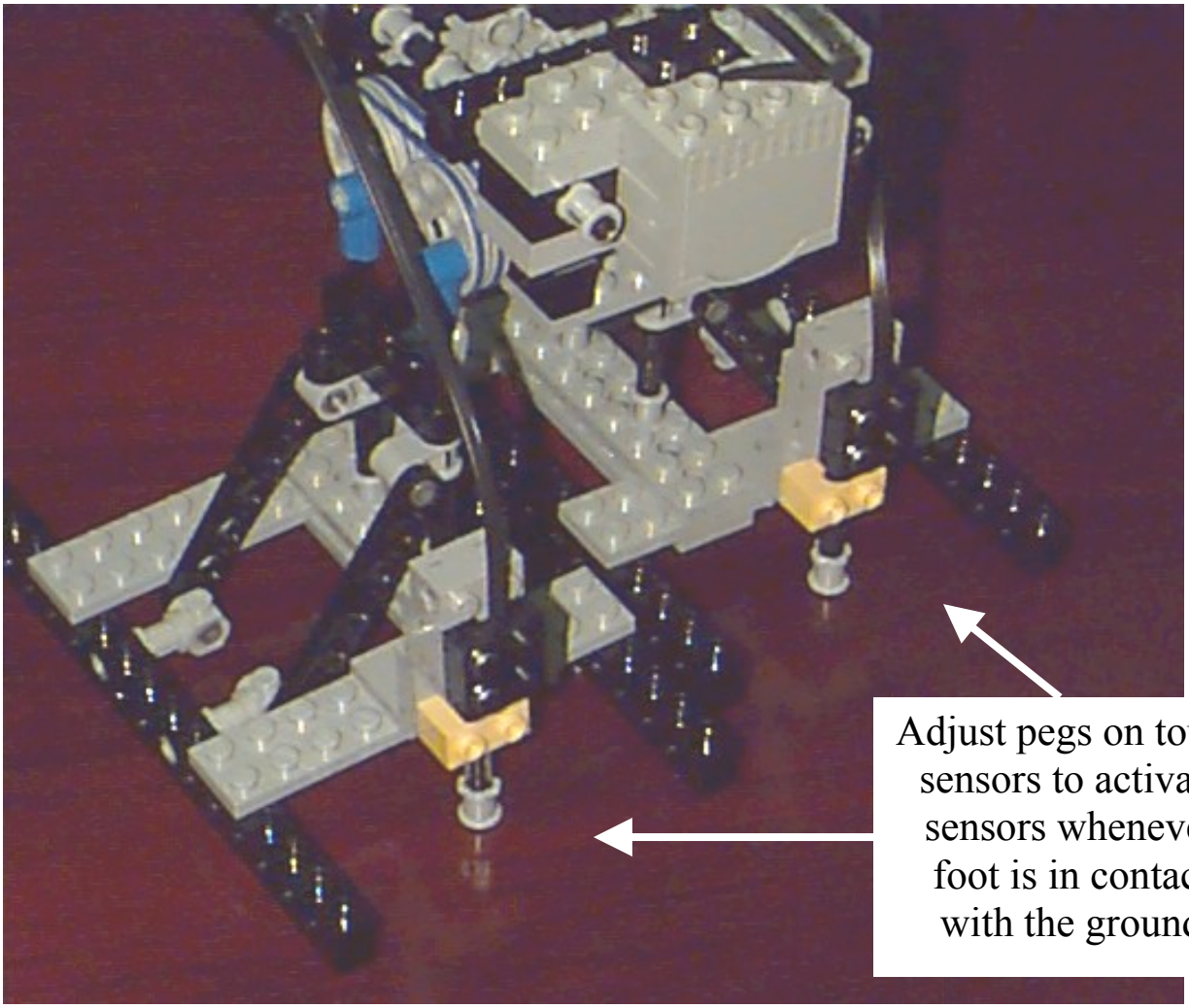
Assembled hip and waist



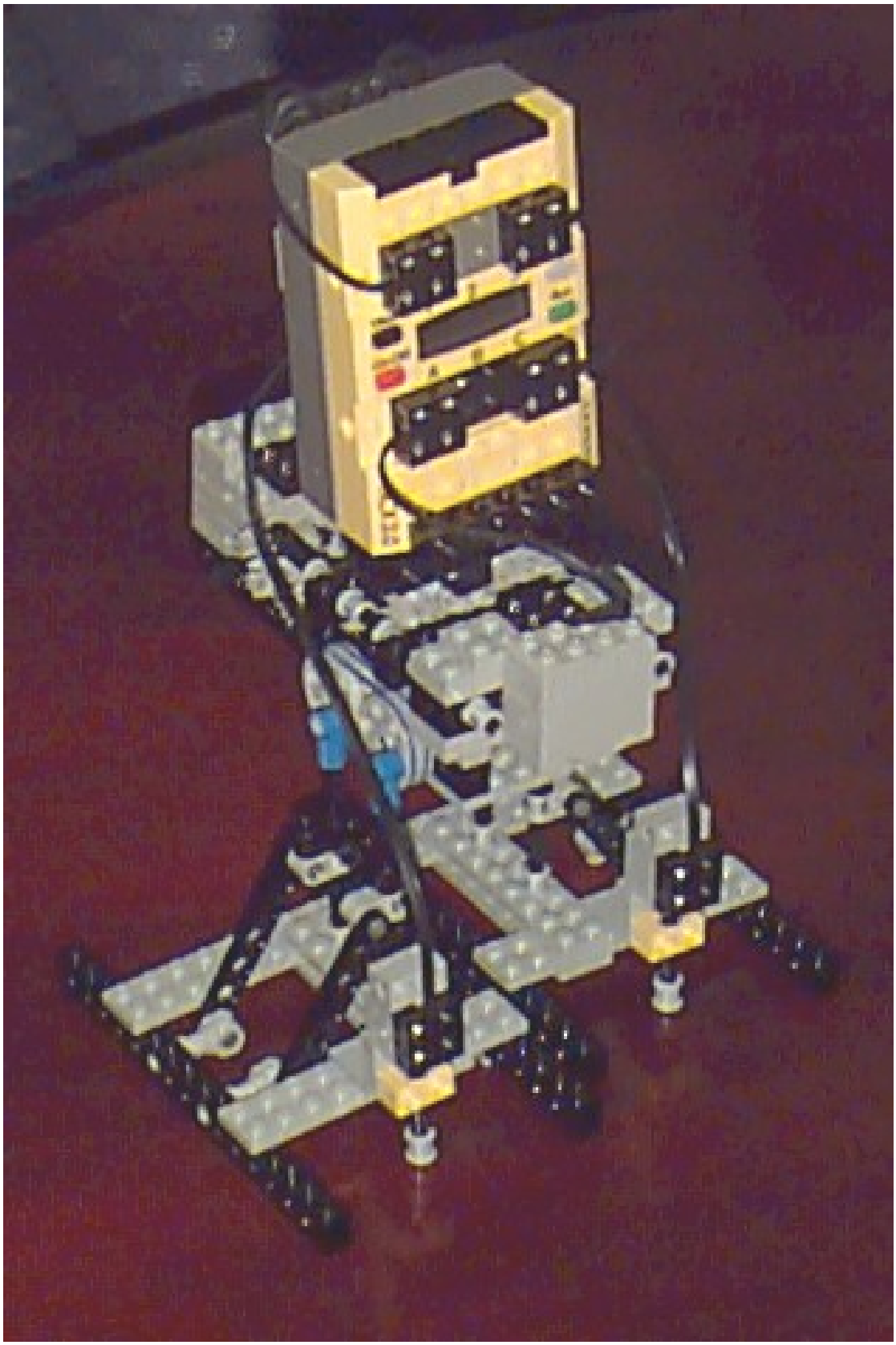
Side view of walker with legs attached



Front view of walker with legs attached



Adjust pegs on touch sensors to activate sensors whenever foot is in contact with the ground



Vita

The author was born on December 31, 1978 in Norfolk, Virginia. He began his undergraduate education at Virginia Tech in 1997 and received bachelor's degrees in Mechanical Engineering and History in December, 2001. On November 23, 2002 Joseph Trout was wed to his wife Jessica LeAnn Smith Trout. Currently, Joseph and Jessica reside in Simmonsville, a very small community in Craig County, Virginia.