Autonomous Convoy Study of Unmanned Ground Vehicles using Visual Snakes

Charles M. Southward II

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

Aerospace Engineering

May 1, 2007

Blacksburg, Virginia

Keywords: Autonomous Convoy, Indirect Visual Servoing

Autonomous Convoy Study of Unmanned Ground Vehicles using Visual Snakes

Charles M. Southward II

(ABSTRACT)

Many applications for unmanned vehicles involve autonomous interaction between two or more craft, and therefore, relative navigation is a key issue to explore. Several high fidelity hardware simulations exist to produce accurate dynamics. However, these simulations are restricted by size, weight, and power needed to operate them. The use of a small Unmanned Ground Vehicle (UGV) for the relative navigation problem is investigated. The UGV has the ability to traverse large ranges over uneven terrain and into varying lighting conditions which has interesting applications to relative navigation.

The basic problem of a vehicle following another is researched and a possible solution explored. Statistical pressure snakes are used to gather relative position data at a specified frequency. A cubic spline is then fit to the relative position data using a least squares algorithm. The spline represents the path on which the lead vehicle has already traversed. Controlling the UGV onto this relative path using a sliding mode control, allows the follow vehicle to avoid the same stationary obstacles the lead vehicle avoided without any other sensor information. The algorithm is run on the UGV hardware with good results. It was able to follow the lead vehicle around a curved course with only centimeter-level position errors. This sets up a firm foundation on which to build a more versatile relative motion platform.[1]

[1]This research was supported by Sandia National Laboratories and Joint Unmanned Systems Test, Experimentation, and Research (JOUSTER). We would also like to thank ORION International Technologies for providing a license to the UMBRA software framework.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many applications for unmanned vehicles involve autonomous interaction between two or more craft. Reliable and robust relative navigation is a key issue. Relative navigation scenarios such as rendezvous and docking of spacecraft, in flight refueling of aircraft, or a convoy of ground vehicles are currently directly managed by humans. Semi-autonomous relative navigation makes these missions operationally simpler and more efficient. Further, the human operator load and exposure to dangerous environments is reduced due to a smaller number of persons needed to direct a multitude of vehicles with high-level directives.

1.1 Visual Path Following

A convoy of ground vehicles moving through an urban setting where each vehicle is currently controlled by a human operator as seen in Figure 1.1. This thesis examines taking the human operators out of the following vehicles. Each follow vehicle would then use visual sensors to follow the vehicle in front of it. This is a highly complex problem where noise in the motion of the vehicle being followed induces larger motion in the follow vehicle. These induced motions may cascade backwards through the convoy creating instabilities. Therefore, as a foundation, the problem of an Unmanned Ground Vehicle (UGV) following a human operated vehicle is investigated first.

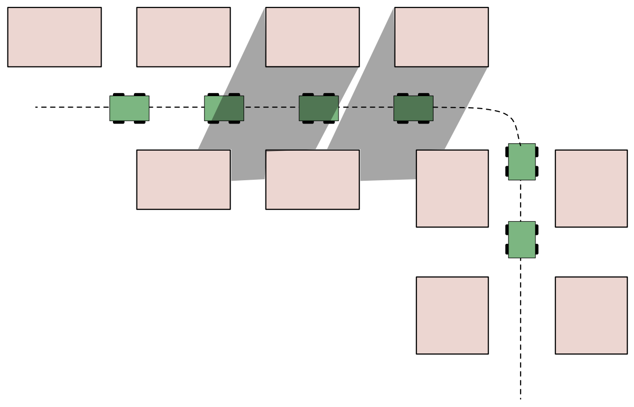


Figure 1.1: Ground Vehicle Convoy Through Urban Terrain.

The figure depicts several other problems that must be addressed in solving this problem. Some high-level challenges that must be addressed include:

- Obstacle avoidance
- Relative position information only
- Robust to lighting conditions
- Variable distances
- No communication between vehicles

Figure 1.1 shows vehicles turning a corner around a building or some other obstacle. Therefore, a solution such as direct servoing would be ineffective because the follow vehicle would cut the corner and impact into the obstacle. Instead of servoing directly onto a visual target, the UGV acquires relative position data of the visual target and constructs a relative path describing the motion of the visual target. Figure 1.2 depicts the discrete relative position data as black dots and shows the relative path that is constructed from the data. The UGV is controlled along this path which allows the UGV to avoid stationary obstacles.

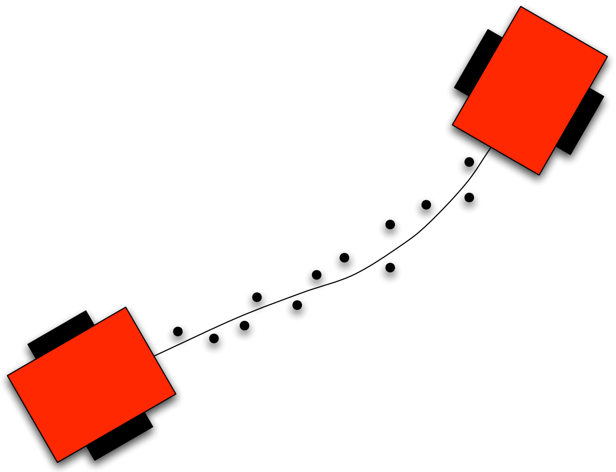


Figure 1.2: Unmanned Ground Vehicle Contructing a Path from Relative Position Data.

A GPS system can be used to get inertial position to control the follow vehicle; however, the relative distances between vehicles may be smaller than the error in the GPS receiver. The follow vehicle might then be controlled into an obstacle. Differential GPS has a much smaller error but still has problems in cluttered environments such as an urban setting. The signal can bounce off several surfaces and create larger errors in the inertial position estimate. More so, GPS does not work without line of sight to the satellites which is a concern in a city landscape. The GPS signal is also susceptible to being jammed. The path following method described only requires relative position information. Accurately determining the relative position then becomes the new challenge. Laser sensors have time of flight problems within close distances, and also emit a signal which could be tracked. Therefore, a new method using statistical pressure snakes is developed to determine the relative position.

The method of statistical pressure snakes uses a camera to gather the required information.

Cameras are sensitive to lighting conditions in that the color of the image appears differently. As seen in Figure 1.1, harsh lighting conditions may be encountered, therefore, the sensor should be robust to varying lighting intensities. Another problem introduced by the operator is a variable desired separation distance. The solution presented must work for a user specified follow distance. This problem is dependent on the resolution of the camera used for the statistical pressure snakes. The higher the resolution, the easier it is to determine the statistical snake information. The follow distance also allows for moving obstacles to pass through the camera view disrupting sensor data, or even stationary obstacles as the lead vehicle turns a corner.

A solution that involves no communication between vehicles has many desirable attributes. Military applications want a minimal amounts of signals to be transmitted by the vehicles which could be intercepted. The signal could be tracked making it easy to locate the vehicles. Communication could make a more robust system. However, the communication hardware can create problems of its own in the form of complexity and computing power. This is coupled with the previous problems as well; if communication is allowed between the vehicles the problem of obtaining relative position is simplified.

1.2 Long Term Implications

The vehicle following problem presented in the previous section will be expanded into several areas of interest. First, as stated, the full convoy problem will be interesting to dissect. How non-smooth motion in the first follow vehicle effects the ability of the second follow vehicle to track it can be evaluated. Another area is aircraft and spacecraft relative motion. The UGV can be used to mimic the dynamics of aircraft and spacecraft in planar motion. The addition of attitude dynamics, from hardware in a lab such as the Space Systems Simulation Laboratory[2] (SSSL), would allow for full three dimensional motion to be simulated.

The problem of relative navigation can be investigated using a hybrid environment in which hardware reproduces the physical motion and software modules simulate the actuation and local dynamics. Figure 1.3 depicts a scenario where a small robot is maneuvering about the International Space Station(ISS). A computer software module would be used to simulate the robot's and the ISS's orbital dynamics and orientation accurately, while an unmanned vehicle with a motion platform can physically implement this relative motion for the visual sensor in indoor or outdoor test environments. Using this flexible hybrid setup, the implementation of a visual relative navigation system in space can be studied.

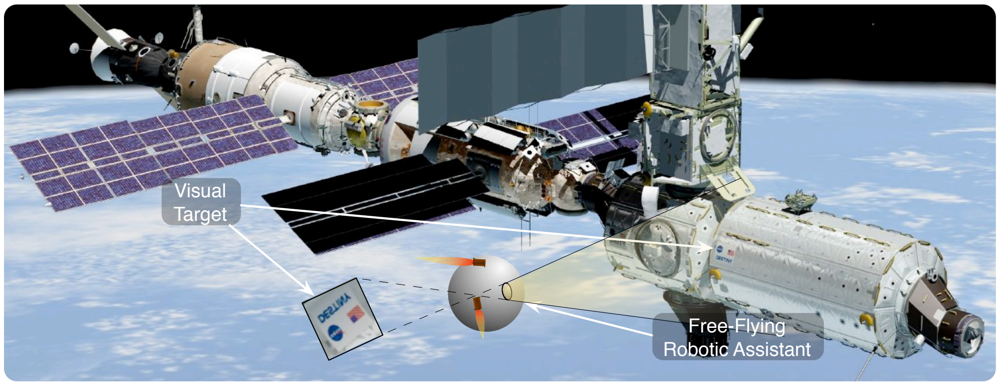


Figure 1.3: Spacecraft using Relative Navigation to Maneuver about the International Space Station[1].

Currently, the Autonomous Vehicle Systems Lab at Virginia Tech includes a UGV and various sensors. The lab has several means of calculating relative position including a passive video segmentation algorithm called Visual Snakes. This calculates the image contour of an object in real time and allows the computation of relative motion information. Dead reckoning is the simplest form of position determination in which the UGV's wheel encoders are used to calculate the change in inertial position. These relative motion sensing tools are currently under development, as well as simulation models of the sensors. They will become part of a hybrid solution to assist us to simulate and research the complex problem of relative navigation.

1.3 Literature Review

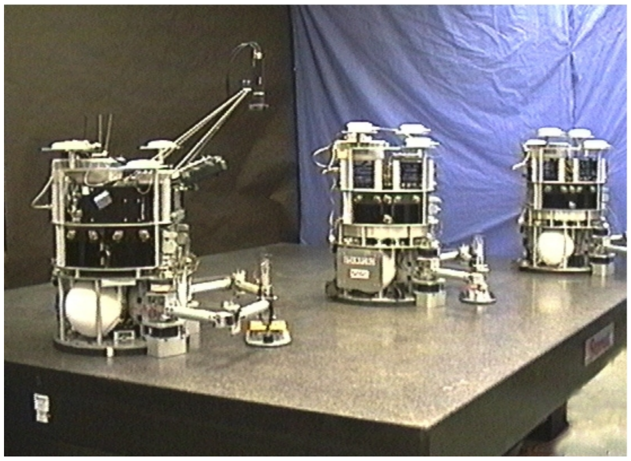


Figure 1.4: Stanford University's Hover Table Testbed.

Since a long term goal is to create a relative motion testbed for unmanned aerospace vehicles, a few other methods are explored. Three methods are widely used to simulate the relative motion of vehicles in space: hover tables, gantry cranes, and pools of water. Stanford University's Aerospace Robotics Laboratory[3] features three autonomous self-constrained free-flying robots shown in Figure 1.4. They use cold-gas thrusters to move about in their seemingly frictionless environment. The three robots are used to test formation flight sensing, planning, and control research. However, the testbed has many restrictions such as the small granite slab that they are constrained to move upon. Because of the simulators size and weight, it cannot be easily moved indoors and outdoors thus restricting tests on visual sensor robustness.

The Navy Research Laboratory has a dual-platform spacecraft Dynamic Motion Simulator,[4] shown in Figure 1.5. It was built to hold full scale models to test sensor and control technologies in six degrees of freedom. Because it is built to manipulate full size spacecraft, the equipment is large and immovable which restricts its use. Therefore, no outdoor lighting conditions can be tested at this facility.

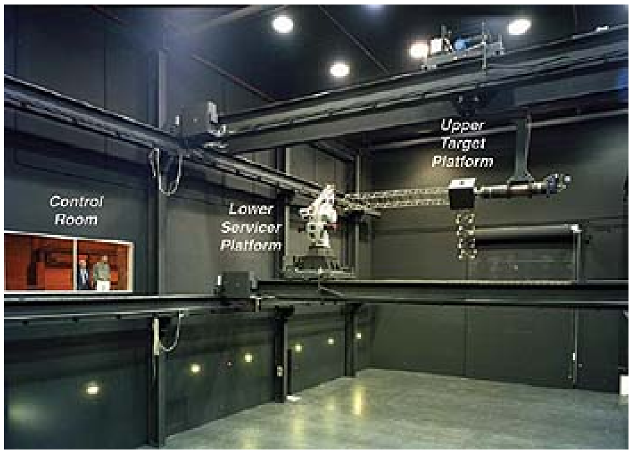


Figure 1.5: Navy Research Laboratory's 6 DOF Dynamic Motion Simulator.

The goal of the AVS lab is to create a hardware testbed simulator like Stanford's or the Navy's with the exception that we want to keep the full simulator small and mobile. The AVS lab uses a UGV from MobileRobots[5] as the simulator platform. An advantage of the small UGV is that it can be used indoors or outdoors with a variety of lighting conditions to stress the visual sensors. Its small size also allows it to cover larger ranges on the order of hundreds of meters, unlike gantry cranes or hover tables.

Exploration into using the UGV as a motion simulator testbed in the AVS lab began with work by Mark Monda[6] and Chris Romanelli.[7] Monda's work uses a digital camera with a statistical pressure snake algorithm to determine the relative position and heading of a target. Non-linear proportional and proportional-integral feedback control laws are developed for the direct visual servoing problem. Direct visual servoing controls the UGV so that the visual target remains in the center of the camera image and a user specified distance from the visual target. The method in this thesis is indirect visual servoing which implies that it is not controlled directly towards the visual target but instead follows a path defined by the visual target position over time. The direct method has the benefit that the visual target is nominally directly in front of the camera and a specified distance away, so that the sensor data is most accurate. However, direct servoing requires relative position data to be obtained at a high rate meaning a large amount of the processor is being used by image processing. Relative position data can be gathered at a fraction of the rate for indirect servoing because the relative position is being used to define a path instead of being controlled upon directly. The visual target can move out of view of the camera or behind an obstacle which results in problems. The indirect method would still follow a path around this target, but the direct servoing method cannot avoid obstacles.

Chris Romanelli's research focused on a software simulation of the UGV. However, the simulation was used to explore controlling the UGV along an orbit path. The UGV simulation was used extensively in the servoing method developed in this thesis. Testing the algorithms on the simulated UGV simplified the problem by not having to deal with hardware issues until the control is ready to be implemented. The simulated and hardware UGV can be run together so that they interact with each other. Romanelli uses Lyapunov's direct method to develop a control to drive the simulated UGV along an orbit path. The orbit path is created by numerical integration of the orbital equations of motion which creates a smooth path. His work does not consider a non-smooth path inherent in the lead vehicle's motion. The indirect visual servo path following uses a sliding mode based control strategy to help account for non-smooth paths.

Kehtarnavaz et al developed another solution to the vehicle following problem as shown in Reference 8. Relative position and heading is obtained by visually identifying a unique tracking feature on the lead vehicle. The work more closely resembles Monda's work where the follow vehicle is being controlled by direct servoing methods. This solution was tested successfully at speeds of 20mi/h at 50ft following distances. Similar work is also done by

Schneiderman[9] where they visually track a target on the lead vehicle to get relative position data. The relative position is then directly controlled upon to move the follow vehicle. The method worked with good results over a 20 mile long distance with varying lighting conditions, curves, and hills. Both methods require fast processing of the relative position data to be successful, which means fast image processing and large processor loads.

A method developed by Gaspar[10] tracks visual landmarks in bird's eye views of the ground plane. The method requires the exact inertial path to be followed, unlike the indirect visual servoing method. The path is then followed by feature detection of environmental landmarks, which results in accurate and precise path following. This method was developed for accurate maneuvers such as docking, where as the indirect servoing method attempts to follow the lead vehicle with minimal error but extreme accuracy is not required.

Das[11] et al develop a vision-based control using feature points of the lead vehicle to estimate velocity and orientation. An extended Kalman filter is used to estimate the linear and angular velocity components as well as the relative orientation. Additionally, the implementation uses an extended Kalman filter to match landmark observations to an a priori map of landmark locations. If an observed landmark is successfully matched, the vehicle's position and orientation are updated. The average error for a test run was approximately 2cm, and few landmarks were seen by the robot. The indirect servoing method requires no a priori data. Additionally the image processing is performed on a desktop computer which is needed in order to process all of the image data.

The next chapter discusses the UMBRA simulation framework and the hardware and visual sensors used in this relative navigation solution. A chapter on the statistical pressure snakes discusses their origin and how they work, as well as how they must be implemented to produce accurate relative navigation. Then the path planner is discussed, closely describing how the relative position is used to fit a cubic spline. The spline is used by the control law. The UGV is controlled along the relative path spline using a sliding mode control. The relative navigation solution is applied to the real hardware and run through two test cases. One test case is a simple fixed visual target; the second is following a visual target along a curved path. Finally, conclusions are drawn about the effectiveness of the solution and what particular features need improvement.

Chapter 2

Ground Vehicle Simulation Description

2.1 The UMBRA Simulation Framework

The UMBRA simulation framework was developed by Sandia National Labs to create high fidelity simulations of complex systems. The framework is based on C++, Tcl/Tk, and an OpenGL graphics environment. The power of UMBRA is the ability to break the complex system up into simpler and smaller components. These simple parts are written into modules using C++. During run time, the aggregate of simple modules relay information to each other and together they form the full complex system. The modules can then be rewritten or improved upon without changing any other modules. Additionally, the modules can be used in other complex systems by only knowing what information needs to be fed to the module. Another flexibility of the UMBRA simulation framework is the integration of Tcl/Tk into the C++ code. During run time, variables in the C++ code can be changed on the fly in the Tcl/Tk command prompt. Therefore, variables such as control gains can be changed during a simulation depending on the current objective.

Virginia Tech's Autonomous Vehicle Systems (AVS) Lab uses the UMBRA simulation framework to simulate complex dynamical systems as well as using actual hardware in conjunction with the UMBRA framework. The hardware in the AVS lab currently consists of an unmanned ground vehicle (UGV), a pan and tilt unit, and frame-grabber card as well as UMBRA modules that implement the visual snake and visual servo control algorithms. Chris Romanelli[7] wrote an UMBRA simulation of the UGV using the full dynamic equations of motion. The UGV simulation is written such that it requires the exact same inputs the actual hardware requires, therefore the hardware and the simulation can be interchanged easily. Tests can be run using the freedom of the simulated UGV with no hardware issues, and later the hardware is quickly swapped with the simulation. This methodology is used in developing the modules covered in this thesis.

The flowchart depicted in Figure 2.1 shows the high-level UMBRA modules created to solve the indirect visual servoing problem. Specifically, the three modules developed are the interface, path planner, and control. Visual snakes gather information from the visual target on the leader vehicle, which is processed by the interface module to get relative position information. The relative position information is sent to the path planner, which determines the desired position of the follow vehicle. The control sends the desired wheel speed commands to the follow vehicle. The loop is closed by sending wheel encoder data to both the path planner and control.

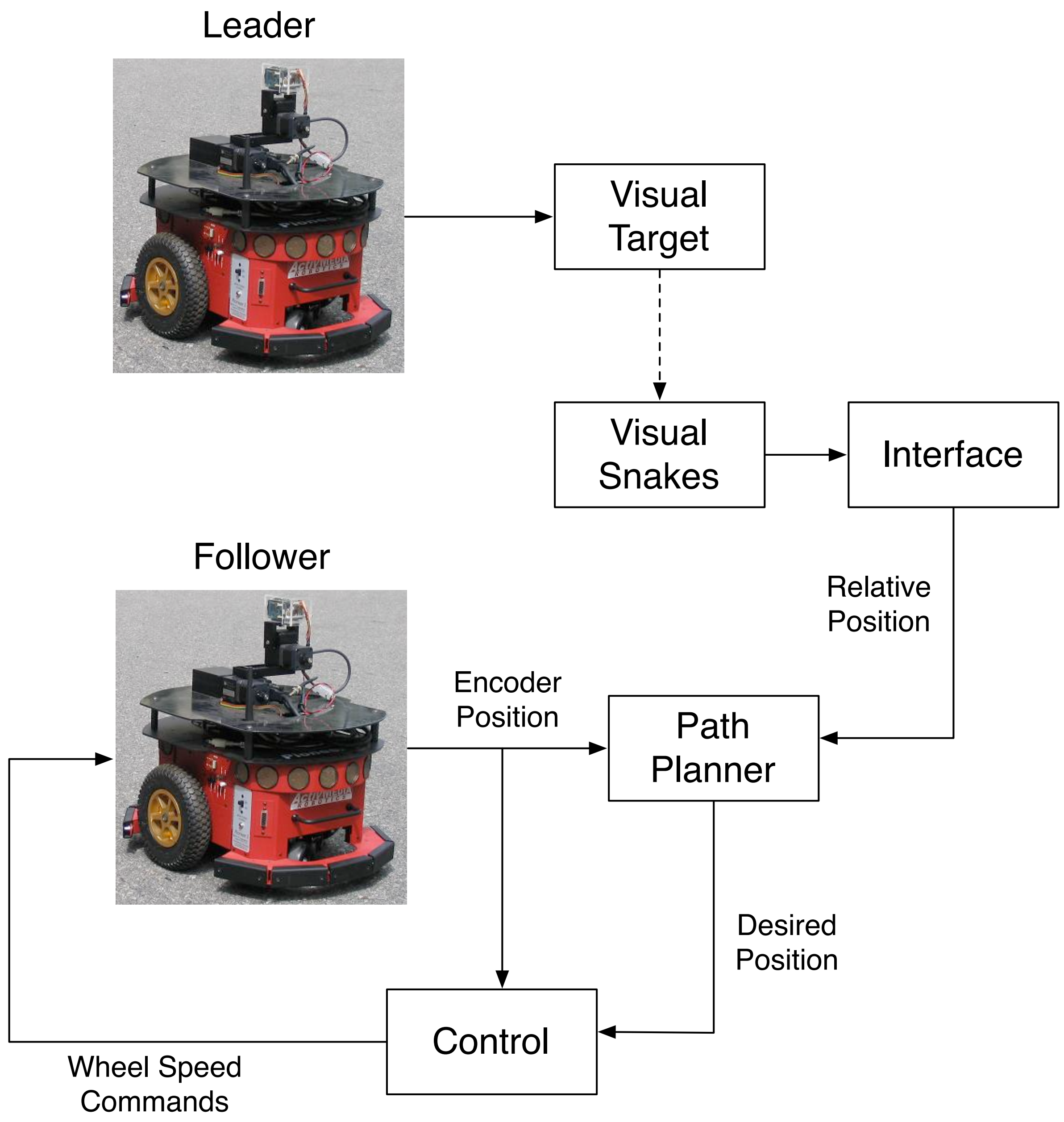


Figure 2.1: High-Level Flowchart of the Required UMBRA Modules.

2.2 Simulation Hardware

2.2.1 Unmanned Ground Vehicle

The AVS Lab's relative navigation testbed relies on an unmanned ground vehicle created by MobileRobots Inc. (formerly ActivMedia Robotics) called Pioneer 3-DX[5] shown in Figure 2.2.1. Pioneer has three wheels, two of which have independent drives and the third wheel for support. A serial interface allows Pioneer to be connected into a PC104 card or external computer. Pioneer has several sensors including wheel encoders, sonar sensors, bump sensors, and a digital three-axis compass/roll/pitch magnetometer. Reference 12 goes into more detail about the capabilities and performance of Pioneer.

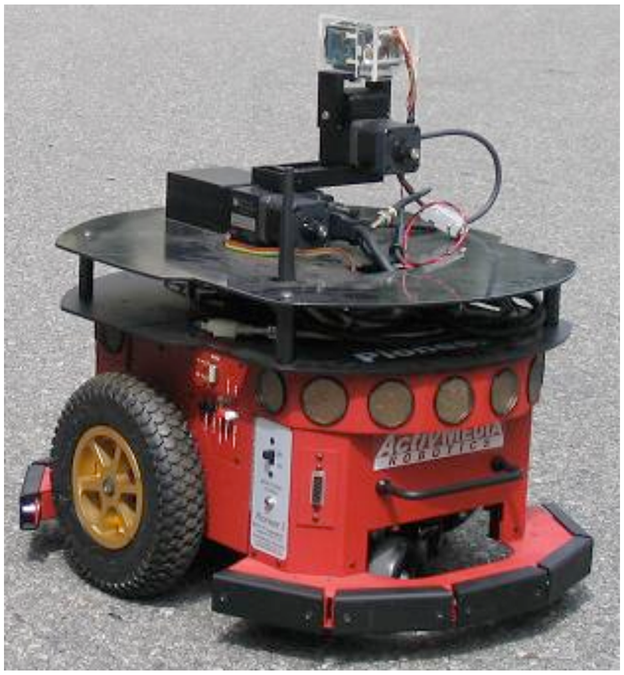


Figure 2.2: MobileRobots Pioneer 3-DX Ground Vehicle with Digital Camera Mounted on a PTU.

The PC-104 computer is based on a Pentium III chip running at 800 Mhz. Currently, it has wireless internet, a frame-grabber card, and a firewire card. The wireless internet connection makes it easy to transfer data from the PC-104 to other computers on the local network, and also allows it to be run remotely. The PC-104 form factor is widely used. Many computer cards can be found to work on it allowing additional capabilities to be added to Pioneer.

Two coordinate frames are used to describe the position and motion of Pioneer. They need to be explained in order to understand descriptions and notations used later in this report. The body frame B axis rotates and translates with Pioneer. The B frame is used to describe the relative position data from onboard sensors. A camera is attached to the Pioneer as well. But for the implementation in this thesis, the camera is fixed in position and rotation so that it does not vary relative to the body frame. The other frame is the inertial frame N. The relative sensor data is mapped into the inertial frame for the implementation in this paper. As well, Pioneer's position and heading is given relative to the inertial frame. The heading angle, θ, is defined as the angle from $\hat{n}_2$ to $\hat{b}_2$. The position, x and y, is defined to the origin of the B frame.

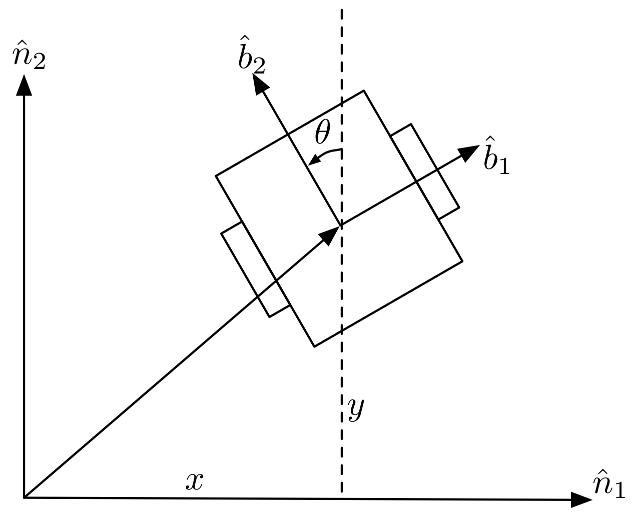


Figure 2.3: The Coordinate Frames and State Variables.

An UMBRA module, called "Pioneer", was written by Mark Monda[6] that controls the robot using the ARIA vehicle control libraries provided by MobileRobots.[5] The UMBRA code was made to work with the ARIA code without the need to customize the ARIA code. Chris Romanelli created a software simulation of the Pioneer UGV, called "Pioneer_sim", that uses a simulated version of the ARIA libraries and requires the same inputs and outputs as the actual Pioneer Hardware. Both Pioneer modules receive a forward speed command and turning speed command. These two commands are then interpreted by the onboard computer to determine the torques required on the wheel motors to produce the desired result. Both modules output inertial position and heading using the wheel encoders. The inertial frame is initialized to line up with the body frame when Pioneer is turned on. Since the simulation and hardware use the same control libraries, the two UMBRA UGV modules produce comparable results and can be interchanged easily.

2.2.2 Sensor Components

A digital camera is mounted on top of the UGV. The camera has a resolution of 640x480 pixels and runs at 30Hz. A frame grabber gets the image from the digital camera and passes it to the statistical pressure snake[13] algorithm, which are then used to find the relative distance to a specified target in the image frame. An overview of the statistical pressure snakes is given in Section 3.1, and an in-depth discussion can be found in References 13.

Each of Pioneer's wheel drive systems uses a high-resolution optical quadrature shaft encoder for precise position sensing and advanced dead-reckoning. There are $36,300$ encoder ticks for one revolution of Pioneer's wheels, or 132 ticks for every millimeter the wheels traverse.[12] The high resolution of the encoders allows the differential inertial motion of the Pioneer to be known precisely. Although the encoders cannot account for issues such as wheel slipping and how well the wheel's radius are known. Therefore, the encoder computed inertial positions will be incorrect after a time span. However, the true inertial position is not required in this control solution as seen in Figure 2.2.2. Even though the follow vehicle does not know its inertial position correctly, the relative path is created correctly because the sensor data is collected from the follower and leader's true positions. The encoder-based inertial motion is only used to measure how far the follower vehicle has traveled between visual sensor updates.

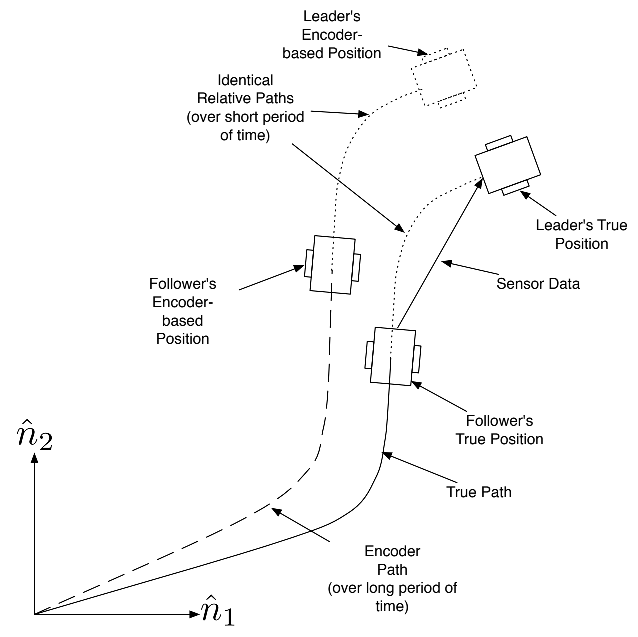

Leader's Encoder-based Position
Identical Relative Paths (over short period of time)
Leader's True Position
Sensor Data
Follower's Encoder-based Position
Follower's True Position
$\hat{n}_2$
True Path
Encoder Path (over long period of time)
$\hat{n}_1$

Figure 2.4: Relative Path Created Regardless of Inertial Position Errors.

Chapter 3

Relative Position Acquisition

The primary sensor measure needed is the position of the target vehicle relative to the following vehicle. This information can be obtained using many different sensors, but the AVS Lab's focus has been on using statistical pressure snakes, or simply visual snakes. The visual snakes gather the relative position data at a specified frequency which creates a path of digital breadcrumbs. Figure 3.1 shows an example of actual breadcrumbs produced by the snakes plotted relative to an inertial coordinate frame. The snakes clearly have noise associated with them, but the path of the lead vehicle can be discerned by the human eye. Because of the noise, the breadcrumbs are later used by the path planner to create a smoothed path for the controller.

3.1 Statistical Pressure Snakes

Statistical pressure snakes are used to find the contour of a desired visual target, as seen in Figure 3.2. The contour is divided into discrete points called control points. The control points are the actual output of the visual snake algorithm. The contour is represented as a

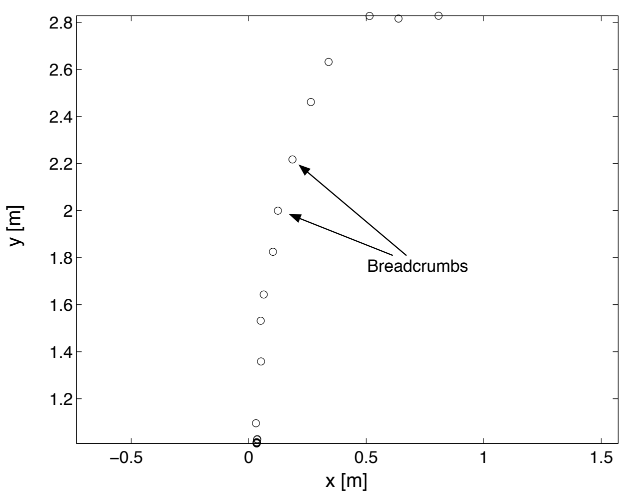


Figure 3.1: Example of Breadcrumbs Produced by Statistical Pressure Snakes.

parametric spline defined in the image space of the camera as

$$S(u) = I(x(u), y(u))' \tag{3.1}$$

where u ranges from $[0, 1]$. The control points are allowed to move, thus changing the shape and position of the spline, such that it minimizes an energy function. Therefore the energy is chosen such that the minimum is along the desired target contour. The first energy model was proposed by Kass et al.[14] as

$$E = \int_0^1 \left[E_{\text{int}}(S(u)) + E_{\text{Pressure}}(S(u))\right] du \tag{3.2}$$

where

$$E_{\text{int}} = \frac{\alpha}{2}\left|\frac{\partial}{\partial u}S(u)\right|^2 du + \frac{\beta}{2}\left|\frac{\partial^2}{\partial u^2}S(u)\right|^2 du$$

where α and β are weights. The visual snakes used in the AVS Lab make two modifications to this energy function described in Reference 13. First E_{int} is replaced by a single term that maintains a constant third derivative. This is done to remove conflicts with the original desired characteristics of visual snakes.[15] Second, the E_{Pressure} is written using a statistical

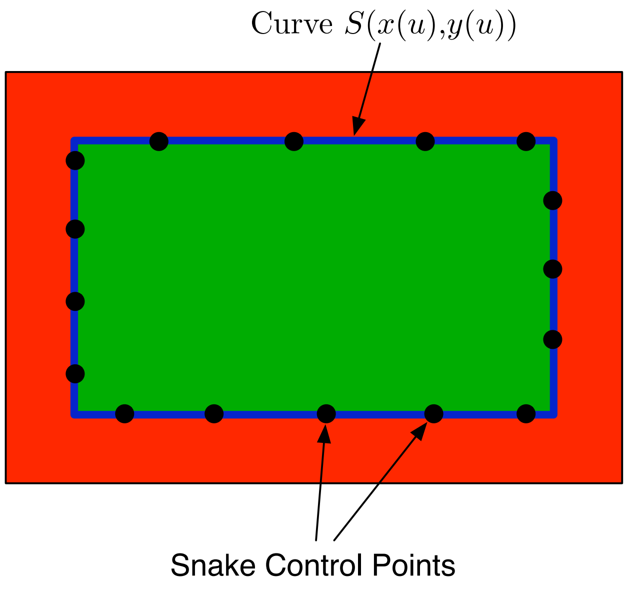


Figure 3.2: Graphical Depiction of an Ideal Visual Snake Contour.

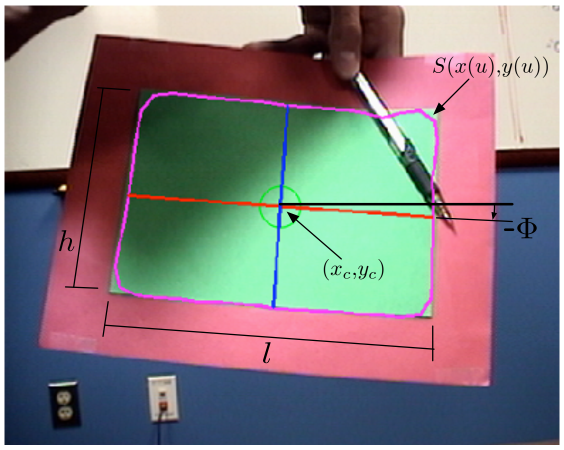
pressure term suggested by Schaub and Smith[13]

$$E_{\text{Pressure}} = \rho \left(\frac{\partial S}{\partial u}\right)^{\perp} (\varepsilon - 1) \tag{3.3}$$

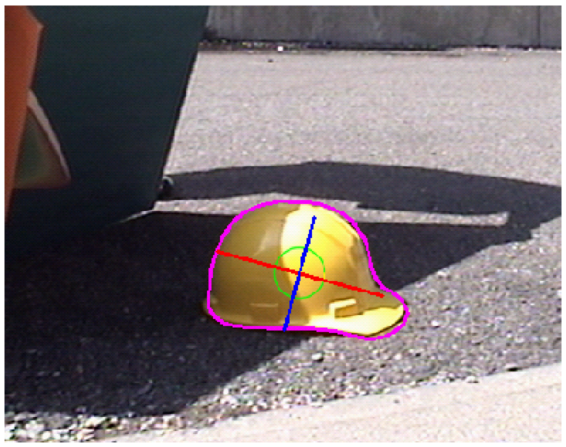
where

$$\varepsilon = \sqrt{\left(\frac{p_1 - \tau_1}{k_1\sigma_1}\right)^2 + \left(\frac{p_2 - \tau_2}{k_2\sigma_2}\right)^2 + \left(\frac{p_3 - \tau_3}{k_3\sigma_3}\right)^2} \tag{3.4}$$

where p_i are local average pixel color channel values, τ_i are the target color channel values, σ_i are the target color channel standard deviations, and k_i are user defined gains. An additional change used in the AVS Lab is mapping RGB space to Hue-Saturation-Value (HSV) color space. By selecting appropriate gains, the visual snakes can track targets with large variations in saturation and shading. For example, one gain selection used in the AVS Lab is referred to as "Hue Only" mode where the gain for Hue is set low and the gains for saturation and value are set to infinite values. The numerical value of hue does not change with brighter lights or darker shadows making it robust to changing lighting conditions shown in Figure 3.3(b). However, similar shades of a color may have the same hue creating problems in unstructured environments where background objects may have similar hues to the target hue. Because the target is chosen by the user, a simple solution is to wrap the main target with a colored border that has a distinctly different hue seen in Figure 3.3(a). The same figure also shows robustness to an obscured target image. Other methods such as straight-line detection would have a tough time discerning the full target contour. Also note that the snake contour does not match the edges perfectly like in the ideal snake contour, shown in Figure 3.2. Because a finite number of control points are used, the corners cannot be tracked perfectly. However, the values of l, h, and Φ are needed to determine the relative position, and the round-off on the corners marginally effects these values.



(a) Visual Snakes Tracking Partially Obscured Target



(b) Visual Snakes Tracking Abnormal Shape in Harsh Lighting Conditions

Figure 3.3: Examples of Visual Snakes Tracking Objects

3.2 Extracting Relative Position from the Statistical Pressure Snakes

Now that the control points of the desired target's contour are found, information such as center of mass (COM), major and minor axes, and the angle between major axis and the horizontal plane of the image can be extracted. To determine this useful information, the moment of the parametric curve is found:

$$M_{ij} = \iint x^i y^j dx\ dy \tag{3.5}$$

where the integral is taken over the area defined by the closed snake contour. This is discretized as

$$M_{ij} = \sum_{n=1}^{N} \sum_{m=1}^{M} x_n^i y_m^j \tag{3.6}$$

where N and M are the number of pixels enclosed by the snake contour along the length, l, and height, h, respectively, as seen in Figure 3.3(a). Because the visual snake is a closed contour, Green's theorem is used to simplify the area integral into a single closed loop integral. The area enclosed by the visual snakes is then M_{00} which can be used to find depth. However, the value of M_{00} changes due to yaw, pitch, and roll of the visual target which skews depth estimates.

More information can be extracted from the visual snakes than simply the area. The COM is determined as

$$x_c = \frac{M_{10}}{A} = \frac{1}{A} \iint x\ dx\ dy \tag{3.7a}$$

$$y_c = \frac{M_{01}}{A} = \frac{1}{A} \iint y\ dx\ dy \tag{3.7b}$$

The COM is used to find the relative heading to the visual target.

In order to find the major and minor inertia axes, the second order moments of inertia are required, shown in matrix form as:

$$[I] = \begin{bmatrix} M_{20} & M_{11} \\ M_{11} & M_{02} \end{bmatrix} \tag{3.8}$$

By treating this inertia matrix similar to an inertia matrix for an arbitrary rigid body, the principal axes are determined using eigenvector decomposition. Defining the principal inertias as $I_1 \geq I_2$, the length and height of the target, depicted in Figure 3.3(a), are found with the following equations:

$$l = \sqrt{\frac{3I_1}{A}} \tag{3.9a}$$

$$h = \sqrt{\frac{3I_2}{A}} \tag{3.9b}$$

These equations assume the target is rectangular. The angle that rotates the inertia matrix in Eq. (3.8) to the principal inertias is the principal rotation angle, Φ, shown in Figure 3.3(a). The next section discusses how the COM, length, height, and principal rotation angle are used to gather accurate relative position data of the lead vehicle.

3.2.1 Heading and Depth Calculation using Statistical Pressure Snakes

Two important parameters of the camera must be found to extract accurate relative position data from the visual snake output. A pin-hole camera model is used to develop the parameters shown in Figure 3.4. The first parameter is the focal length, f, of the camera, which is needed to determine the body fixed x-direction component of the relative position. The focal length is calibrated by placing a visual target at user defined coordinates of D and X, which produces the angle θ. The focal length is calculated by using the x_c value of the COM and θ shown as:

$$\mathrm{f} = \frac{x_c}{\tan\theta} \tag{3.10}$$

Now knowing the focal length and measuring x_c, the relative heading angle is determined.

The second parameter is the depth gain, γ, which is found using similar triangles:

$$\frac{D}{X + \frac{L}{2}} = \frac{\mathrm{f}}{x_c + \frac{b}{2}} \tag{3.11}$$

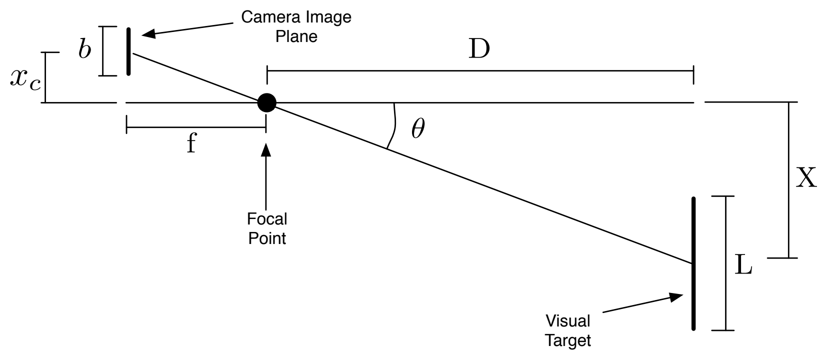


Figure 3.4: Pin-hole Camera Model.

Substituting $X = D\tan(\theta)$ and expanding leads to

$$D\left(x_c + \frac{b}{2}\right) = f\left(D\tan(\theta) + \frac{L}{2}\right) \tag{3.12}$$

Solving for the depth, D, gives

$$D = \frac{\frac{L}{2}f}{x_c + \frac{b}{2} - f\tan(\theta)} \tag{3.13}$$

Substituting for x_c from Eq. (3.10), the depth simplifies to

$$D = \frac{L\text{f}}{b} \tag{3.14}$$

Placing the visual target at a known depth and measuring the value of b, which can be either the length or height, the depth gain parameter is found

$$\gamma = D\ b = L\ f \tag{3.15}$$

The depth to the visual target is now found using the calibrated depth gain value as

$$D = \frac{\gamma}{b} \tag{3.16}$$

The pinhole model works well in estimating the relative position around the nominal depth, D, and angle, θ, chosen when calibrating. Because the vehicle is following the leader at a fixed desired distance, the pinhole model works for this problem. The only detail left is whether to use the height or length of the rectangular target for the value of b in the depth calculation. Also, note that if the follower is required to change the separation distance over time, then this calibration would need to be performed over a range of separation distances.

3.2.2 Implementing the Depth Gain

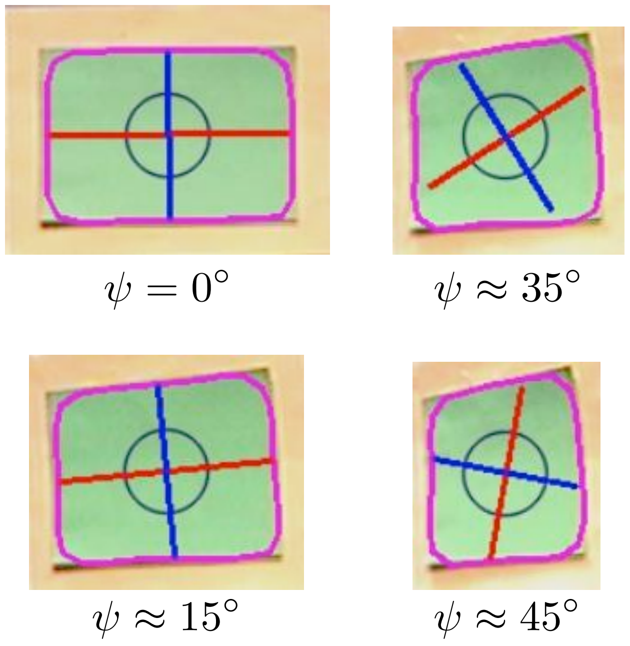
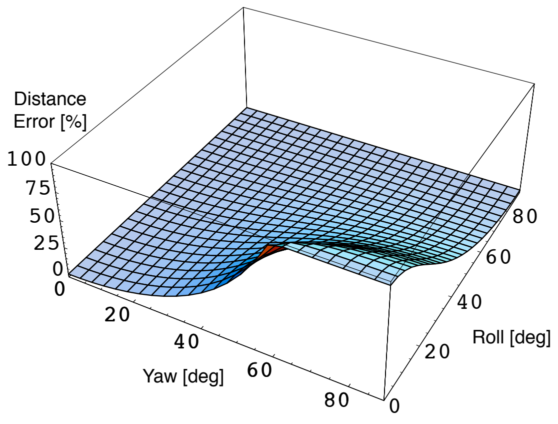


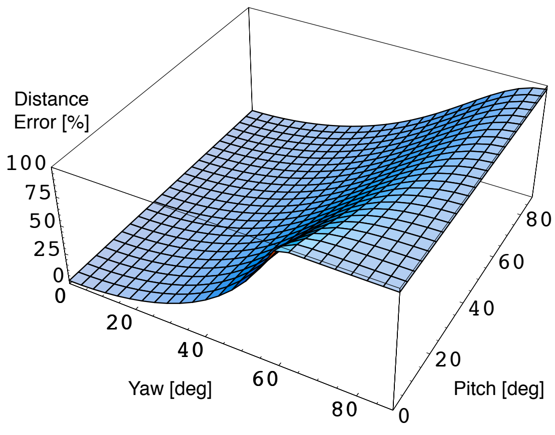
Figure 3.5: Visual Snake Problem when Lead Vehicle Turns.

Mark Monda[6] used the visual snakes by developing a control that moved Pioneer relative to the COM of the visual snakes. His development assumed Pioneer's camera image plane would be parallel to the visual target. However, the approach developed in this paper uses a path planner to control Pioneer; therefore the camera image plane may not always be parallel to the visual target. This problem occurs when the lead vehicle begins to turn, or yaw (ψ), about its third axis as shown in Figure 3.5. The visual targets in the figure have small roll and pitch angles as well. Also, notice that the rectangular target is setup such that the major inertia axis is nominally horizontal. Two problems are seen in this figure, first the major axis (in red) shortens as the lead vehicle begins to turn seen in the $\psi = 0°$ and $\psi \approx 15°$ cases. Imagine the lead vehicle rotating but not translating, if the major axis is used to determine relative distance, it appears as though the lead vehicle is moving away from the camera frame as its length shortens. However, the minor axis (in blue) is along the vertical dimension and remains relatively unchanged. Another problem appears for large yaw angles as shown in Figure 3.5 for yaw angles of $\psi \approx 35°$ and $\psi \approx 45°$ cases. The major and minor axes are rotated by the principal rotation angle Φ so that the minor axis is nominally along the horizontal. The minor axis now experiences the shortening effect as the vehicle continues to turn.

Using the Major Axis for Depth Calculation



(a) Yaw versus Roll

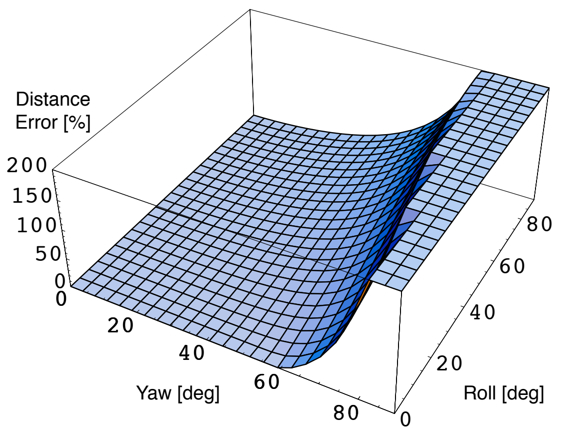


(b) Yaw versus Pitch

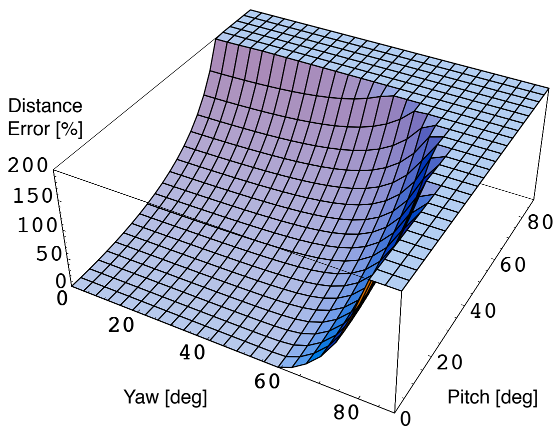
Figure 3.6: Distance Error in Visual Snake Data using Maximum Inertia Axis

A numerical simulation of the visual snakes is done to see how yaw affects the visual snake readings. Additionally, because the vehicle may be used on uneven terrain, the visual snake data was plotted versus roll and pitch as well. However, for the UGV visual tracking application discussed, these roll and pitch angles can be assumed to be small. Figure 3.6 shows results when using the length of the major axis, in place of b in Eq. (3.15), to calibrate depth gain and analyze the visual snake data. As yaw increases with no roll, the error in the relative distance measured by the visual snakes doubles. The error begins to grow immediately due to the shortening effect but notice that it levels out. This is due to the rotation of the major and minor axes, so the major axis is along the height and becomes unchanging with increasing yaw. The roll actually improves the distance estimate because it counters the rotation of the major and minor axes shown in Figure 3.6(a). The pitch does nothing to improve or deteriorate the distance measurement as seen in Figure 3.6(b).

Using the Minor Axis for Depth Calculation



(a) Yaw versus Roll

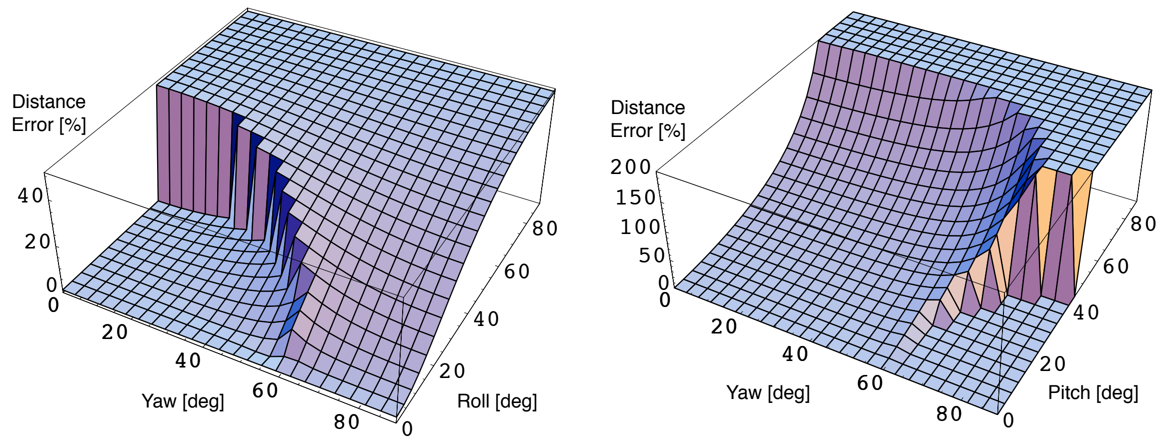


(b) Yaw versus Pitch

Figure 3.7: Distance Error in Visual Snake Data using Minimum Inertia Axis

Errors are present when using the major axis because of the foreshortening problem discussed, therefore the minor axis is examined because it does not experience the foreshortening effect for pure yaw motion. The numerical results are shown in Figure 3.7. As the lead vehicle yaws, the minor axis is unchanged in length so the distance error remains zero. Once the major and minor axes rotate so that the major axis represents the height, the distance error actually grows to infinity because the minor axis now shortens to zero as yaw approaches 90°. As the roll increases, the rotation between major and minor axes occurs sooner, shown in Figure 3.7(a). Pitch has the greatest influence because it creates a shortening effect on the minor axis, similar to how yaw shortens the major axis, shown in Figure 3.7(b).

New Switching Strategy between Major and Minor Axes for Depth Calculation



(a) Yaw versus Roll (b) Yaw versus Pitch

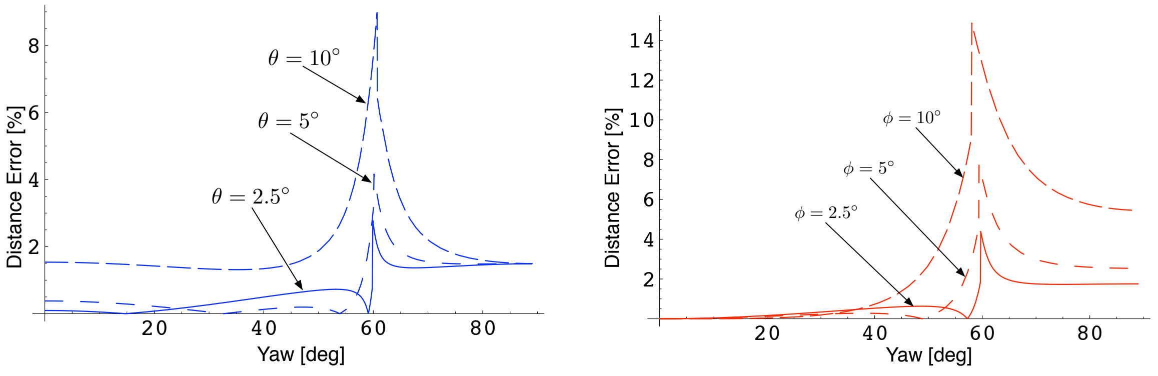
Figure 3.8: Distance Error in Visual Snake Data using a Combination of Max and Min Inertia Axes

Neither axis by itself provides a good result once the major and minor axes rotate. The rotation occurs instantaneously when the principal rotation angle is $\Phi = 45°$ for cases with no roll or pitch. When roll and pitch are present, the rotation of the major and minor axes occurs over a range of Φ but still centered about $\Phi = 45°$. Therefore, switching between the major and minor axes is used to benefit from both. The depth is then found as

$$D = \frac{\gamma}{h} \quad \text{for} \quad \Phi < 45^\circ \tag{3.17a}$$

$$D = \frac{\gamma}{l} \quad \text{for} \quad \Phi \geq 45^\circ \tag{3.17b}$$

where l and h are the length and height of the rectangular target which are estimated using the inertias as seen in Eq. (3.9). Note that the same γ value is used in both cases. This strategy attempts to compute the distance, D, using the target's rectangular vertical axis.

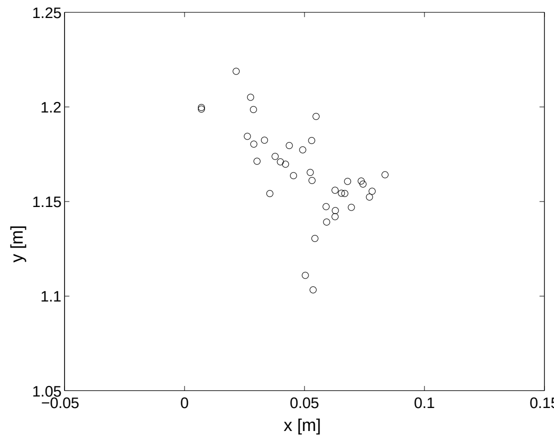


(a) Various Pitch Angles

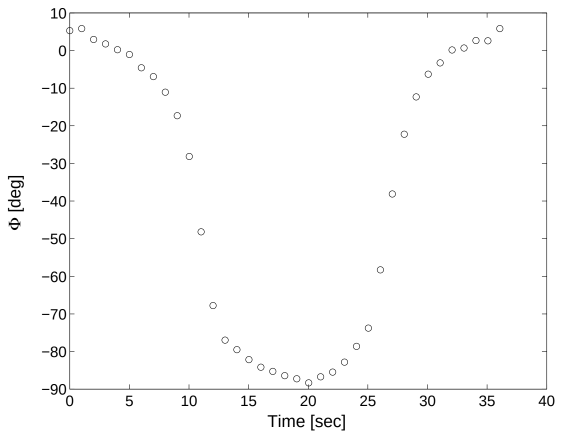
(b) Various Roll Angles

Figure 3.9: Distance Errors for Discrete Pitch and Roll Angles.

The numerical results are shown in Figure 3.8. As yaw increases, the minor axis does not change length so the distance estimate has no error. When the major and minor axes rotate, the switching is implemented. The major axis is now along the vertical dimension and does not change length so the distance estimate remains error free. Roll and pitch are added to the numerical simulation, the error grows slightly before the switching method is implemented. After the switch, the error decreases back to zero. Pitch causes a foreshortening effect, so distance errors increase to infinite values. However, the distance error remains under 10% for pitch angles under 10° as shown in Figure 3.9(a). The distance error is under 15% for roll angles under 10° as shown in Figure 3.9(b).



(a) Inertial Positions of the Extracted Breadcrumbs



(b) Principal Rotation Angle

Figure 3.10: Results from the Implementation of the UMBRA Interface Module using the Switching Method to Extract Relative Position.

The switching method is now tested on the hardware to show if it works in a real application. A visual target was set at a fixed distance away according to the depth gain calibration. The UMBRA interface module is run and the target is slowly rotated about its third axis to roughly 80° yaw and back to zero. Note that the target is rotated by a human so there is most certainly roll and pitch in the data, but only to a small degree. The resulting data is shown in Figure 3.10. The breadcrumbs are plotted in an inertial coordinate frame and appear to clump in a relatively small area with a standard deviation of 2.5 cm, as seen in Figure 3.10(a). Nothing is used to measure the yaw angle, but the principal rotation angle gives an idea of how the visual target is moving, shown in Figure 3.10(b). The principal rotation angle decreases to $-90°$ and back up to zero as the yaw angle is rotated to 80° and back to zero. The switching occurs twice at roughly 10 s and 27 s.

3.3 Interface Module

The UGV relative target position calculation is implemented in an UMBRA module called interface as seen in Figure 2.1. The inputs and outputs to the interface module are shown in Figure 3.11. The output from the visual snakes is taken into the interface module, and the UGV-centric relative position data is the output. The visual snake module has two output connectors, the first passes the visual snake state data (COM, Φ, length, height, and area) with units of pixels and radians. The second is a status flag which tells the interface if there are errors or if the snake is running. The interface implements the new depth sensing discussed and outputs the sensor data and a path status. The sensor data is the relative position of the lead vehicle and has units of meters. The path status is a flag which states if a new breadcrumb has been received. This allows the path planner to recognize the sensor data as a new breadcrumb. The breadcrumbs create a history of where the lead vehicle has been. However, the breadcrumbs have errors associated with them as discussed in this section. The next chapter discusses the path planner module which takes the breadcrumbs and smoothes them out for the control.

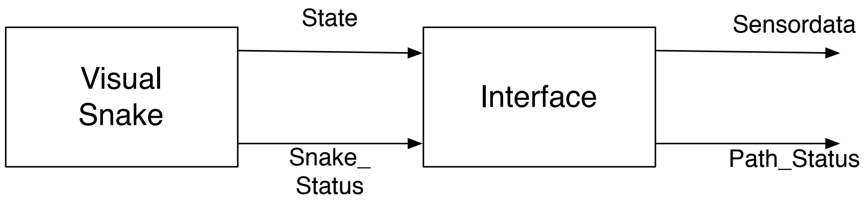


Figure 3.11: Inputs and Outputs to UMBRA Interface Module.

Chapter 4

Path Planner

The indirect visual servoing method requires a smooth path to follow. One reason for a smooth path is that the visual target must remain in the camera field-of-view. An unsmooth path might cause the follow vehicle to lose sight of the visual target. All sensors produce noisy measurements, so the sensor data cannot be used directly as the path. However, the sensor data can be filtered to create a smooth path to follow. The purpose of the path planner is to use the sensor data to create a smooth path for the control. The path planner is also used to find a smooth desired follow speed and heading angle.

The high-level flow chart in Figure 2.1 shows how the path planner module fits into the full indirect visual servoing solution. As is shown in Figure 4.1, where BC is short for breadcrumb, finding a smooth path is only one step in the UMBRA path planner module. The flag _stage defaults to zero, so the module does nothing until the first breadcrumb is received. The path status flag input from the interface module informs the path planner when a breadcrumb is received. Once received, there are two main parts to the path planner, one takes in new breadcrumbs and creates the smooth path, labeled as "Process New Breadcrumbs" in the flow chart. Processing the new breadcrumbs starts with an initialization which is run when the first breadcrumb is received. After initialization, the newest breadcrumb is added to an array of breadcrumbs and old breadcrumbs meeting a certain criteria are removed from the array. A least squares fit is then used on the array of breadcrumbs to define a smooth path. The smooth path is then used to retrieve a desired follow speed and also the desired position. The second part of the path planner module propagates the current desired follower position according to the least squares fit, labeled as “Propagate Along Desired Path”. Given the change in time, the desired target vehicle speed is used to get a distance traveled within that time. The distance traveled is then propagated forward on the smooth path and the desired position is sent to the control module.

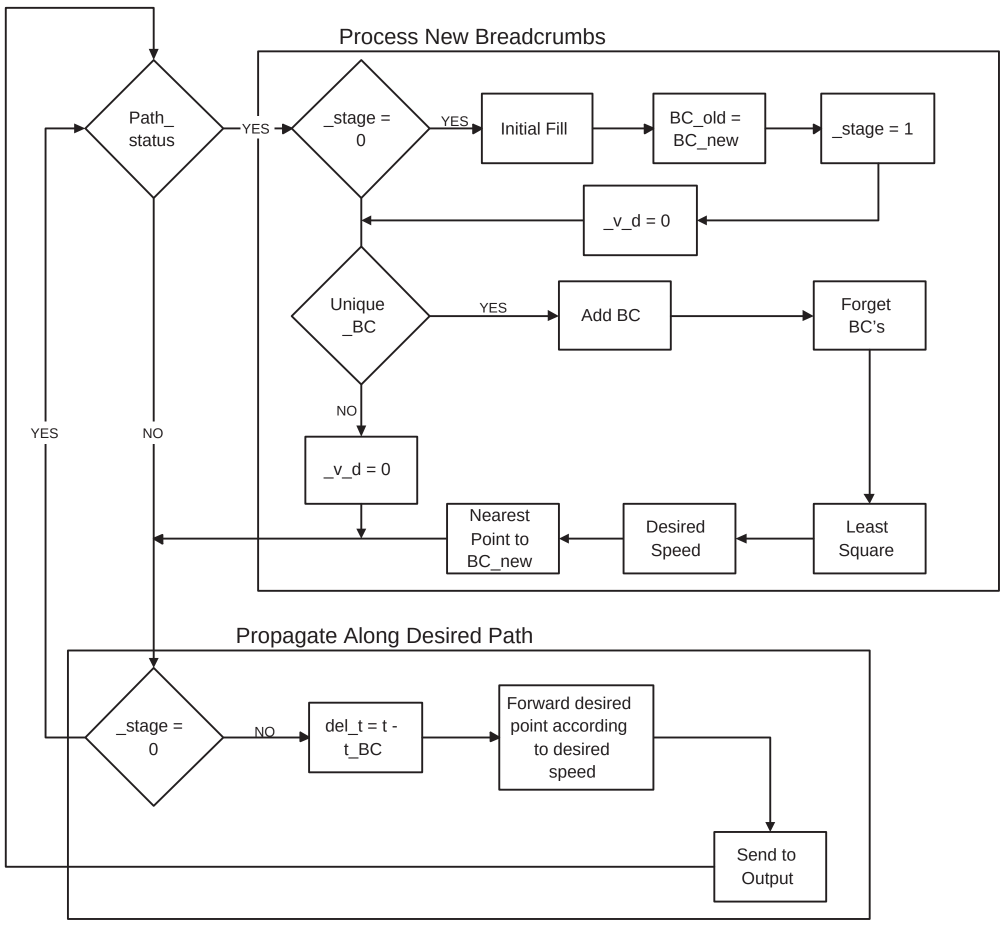


Figure 4.1: Flow Chart of the UMBRA Path Planner Module.

4.1 Processing New Breadcrumbs

4.1.1 Fitting a Cubic Spline

The main purpose of the path planner is to create a smooth relative path of the target vehicle with respect to the follower vehicle. A least squares fit to a cubic spline is used to smooth the sensor data. This is shown as "Least Square" in the flowchart of Figure 4.1. The sensor data needs be known in a common frame, so each new breadcrumb from the visual snake sensor is rotated into the inertial frame N:

$$x = x_p + x_b \cos(\theta) - y_b \sin(\theta) \tag{4.1a}$$

$$y = y_p + x_b \sin(\theta) + y_b \cos(\theta) \tag{4.1b}$$

Here x_p and y_p are the inertial position of the follow vehicle, θ is the heading of the follow vehicle, and x_b and y_b is the relative position received from the sensor in the body frame.

Each breadcrumb received is added to an array of breadcrumbs along with the time associated with the breadcrumb. In order to account for sharp turns that cannot be defined by a function dependent on position, such as $y(x)$, the least squares algorithm is implemented on two parameterized functions, $x(\tau)$ and $y(\tau)$. The variable τ is a non-dimensionalized time where $\tau = 0$ corresponds to the time at the oldest breadcrumb, and $\tau = 1$ corresponds to the time at the newest breadcrumb. The least squares algorithm requires a cubic term of τ, as will be shown. If true time were used, it would grow unbounded and cause numerical precision issues.

The cubic spline fit is:

$$x(\tau) = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 \tag{4.2a}$$

$$y(\tau) = b_0 + b_1\tau + b_2\tau^2 + b_3\tau^3 \tag{4.2b}$$

where the least squares formula solves for the a_i and b_i constants. The breadcrumbs provide the x and τ values, so the a_i constants are solved for as:

$$\begin{Bmatrix} x(\tau_1) \\ x(\tau_2) \\ \vdots \\ x(\tau_n) \end{Bmatrix} = \begin{bmatrix} 1 & \tau_1 & \tau_1^2 & \tau_1^3 \\ 1 & \tau_2 & \tau_2^2 & \tau_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \tau_n & \tau_n^2 & \tau_n^3 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = [M]\,\boldsymbol{a} \tag{4.3}$$

where n is the number of breadcrumbs in the array. There are 4 unknowns and more than 4 equations so this is where the least squares fit comes in. The pseudo-inverse is taken to solve for $\boldsymbol{a}$:

$$\boldsymbol{a} = [M]^{\dagger}\,\boldsymbol{x} \tag{4.4}$$

and the pseudo-inverse is found to be:

$$[M]^{\dagger} = \left([M]^T [M]\right)^{-1} [M]^T \tag{4.5}$$

Using a cubic spline fit, the inverse term in Eq. (4.5) is of a 4x4 matrix. The inverse is solved for symbolically and then directly implemented in C-code in the path planner module.

For the current set of breadcrumbs in the array, the position along the curve is now known using Eq. (4.2). The same method is run to solve for the b_i terms.

4.1.2 Forgetting Prior Breadcrumbs

The least squares algorithm fits the breadcrumbs to a cubic spline. Over a short distance a cubic spline works well. But, as the number of breadcrumbs increases, the path of the lead vehicle may move in a way that a cubic spline does not approximate the motion well. Because the follow vehicle is following at a close distance behind the lead vehicle, the cubic spline need only match the path between the two vehicles. Forgetting past breadcrumbs will

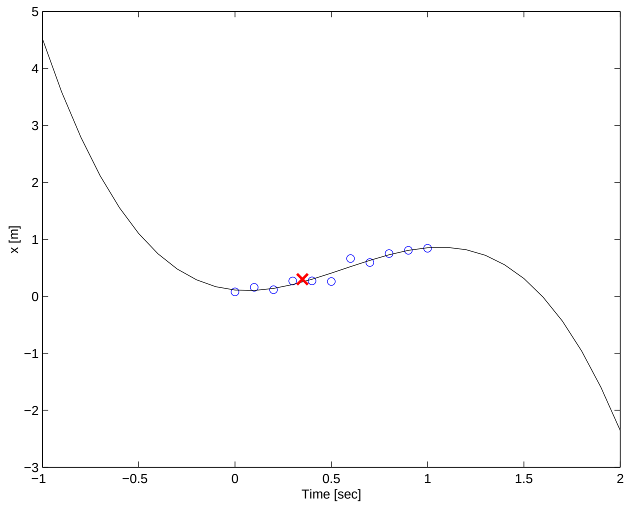


Figure 4.2: Example of Desired Path Veering due to Least Squares Algorithm.

produce a closer match to the local breadcrumbs. The function to forget prior breadcrumbs is shown in the flow chart of Figure 4.1 as “Forget BC’s”.

Past breadcrumbs are forgotten after they pass a certain distance behind the follow vehicle. The reason for keeping points behind the follow vehicle is because the least squares algorithm does well interpolating data but not extrapolating. The path may veer off sharply at the edge of the data points, as seen in Figure 4.2 where the red ‘X’ symbolizes the follow vehicle. The spline fits the data points well between the first and last breadcrumb, but afterwards it turns sharply.

The first step in this calculation is to find the closest breadcrumb to the follow vehicle depicted in Figure 4.3. Starting at the newest breadcrumb, the magnitude of the vector between the follow vehicle’s position and all n number of breadcrumbs is found. This is depicted as the vector $\boldsymbol{\rho}_i$ in the figure, where $i = 1, 2, 3, ..n$. The minimum magnitude found corresponds to the closest breadcrumb to the follow vehicle. Knowing the closest breadcrumb, the distances are now calculated between it and the breadcrumbs occurring prior in time to it, shown as the vector $\boldsymbol{s}_i$. Once the first breadcrumb is found that exceeds

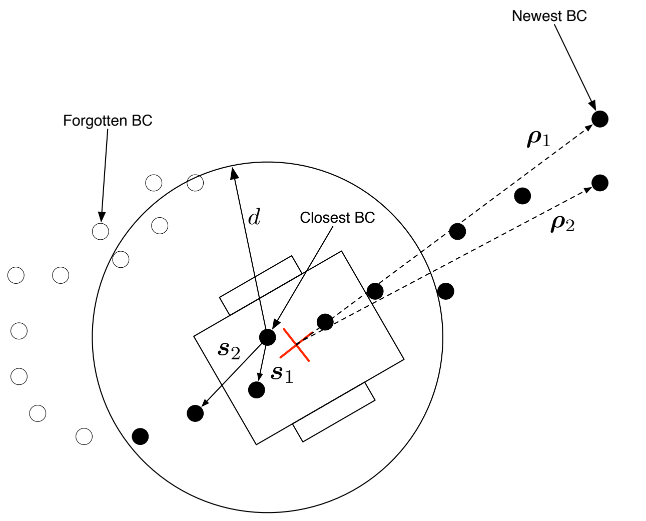


Figure 4.3: Graphical Description of How Breadcrumbs are Forgotten.

the forget distance, all breadcrumbs occurring prior to it are forgotten shown in the figure as open circles. The breadcrumb array is set up as a First-In-First-Out(FIFO) buffer. Instead of erasing the forgotten breadcrumbs and shifting the array, the pointer to the current oldest breadcrumb is updated saving computation time. When the end of the FIFO buffer is reached, the new breadcrumbs are wrapped around to beginning of the buffer again. Even though some of the forgotten breadcrumbs are within the forget distance, d, by working backwards through the breadcrumbs as they appeared in time, even breadcrumbs that are inside d are forgotten.

4.1.3 Desired Following Position and Velocity

The desired distance to be following the lead vehicle is defined by a Tcl/Tk function that is called during run time and can be changed at anytime. The desired following distance may change depending on the application or terrain, so it was designed to be changed easily and on the fly. The process for determining the following position is similar to find the breadcrumbs to be forgotten. However, the cubic spline is now being used instead of the breadcrumbs themselves. A graphical representation is shown in Figure 4.4.

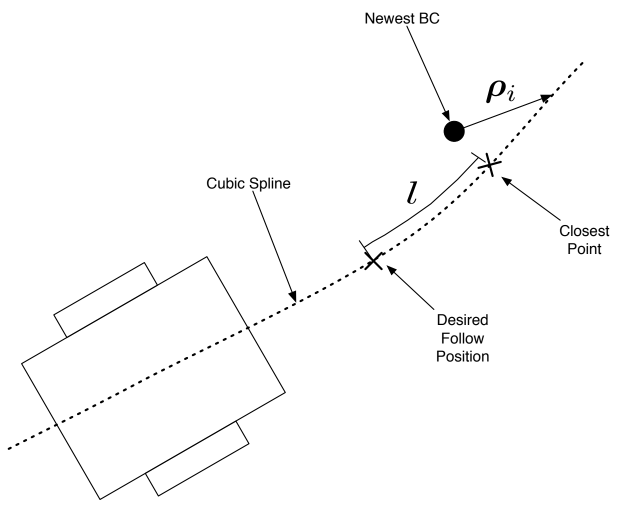


Figure 4.4: Graphical Description of How the Desired Following Position is Calculated.

The cubic spline is divided up into small intervals from $\tau = 0$ to 1.1. Because the nearest point may not occur before $\tau = 1$, the range is extended further. The magnitude of the vector from the newest breadcrumb to each of these points along the cubic spline is found, shown as the vector $\boldsymbol{\rho}_i$ where i represents each point on the cubic spline. The minimum magnitude of $\boldsymbol{\rho}_i$ is therefore the closest point on the cubic spline to the newest breadcrumb. This method is similar to how the closest breadcrumb is found to the follow vehicle (see Eq. (4.8)).

Using the same divisions along the cubic spline and starting at the closest point on the cubic spline just found, the distances are then calculated along each interval of the cubic spline until the following distance, l, is met. This point on the cubic spline is then deemed to be the desired following position, and is the position used in the propagation scheme discussed in the next section.

The last piece of the puzzle is to find the desired velocity of the follow vehicle. Differentiating the breadcrumb positions

$$v = \frac{\sqrt{\left(x\left(t\right) - x\left(t - \Delta t\right)\right)^2 + \left(y\left(t\right) - y\left(t - \Delta t\right)\right)^2}}{\Delta t} \tag{4.6}$$

where x and y are the positions of the breadcrumbs at the current time, t, and the previous time, $t - \Delta t$. However, when differentiating, noise in the data is amplified and since the visual snakes have noise associated with their data, some filtering needs to be done. By substituting the closest points on the cubic spline to the breadcrumbs into Eq. (4.6), a statistical filtering is preformed on the data. Also, the closest point on the cubic spline is already calculated when finding the desired follow distance. By storing this in memory, no extra calculations have to be done to find the closest point again.

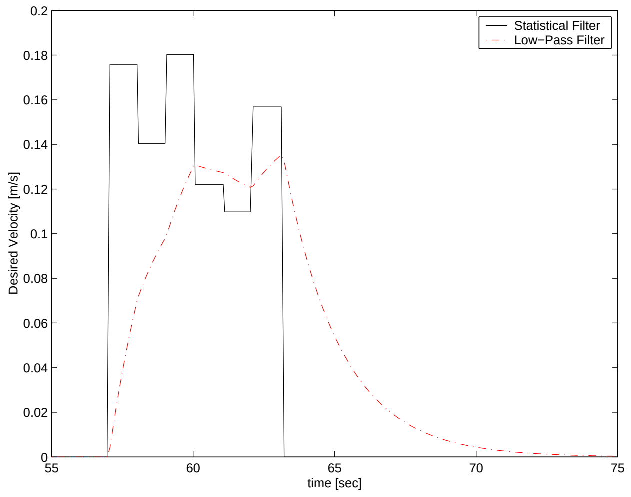


Figure 4.5: Filtering Results on the Desired Velocity.

After implementing the desired velocity in this manner, the values still spiked over time so a first-order low-pass filter is implemented on top of the statistical filter. Note that the statistical filter is implemented only when a new unique breadcrumb is received, but the low-pass filter is actually implemented in the propagating section of the code which occurs every time the module is run. The low-pass filter is of the form:

$$v_f(i) = \frac{1}{2 + \Delta t\omega_c}\left[v_f(i-1)\left(2 - \Delta t\omega_c\right) + \Delta t\omega_c\left(v(i) + v(i-1)\right)\right] \tag{4.7}$$

where v_f is the low-pass filtered velocity, v is the statistically filtered velocity, and ω_c is the cutoff frequency chosen by the user. This requires previous states of the velocity and filtered velocity, so they are initialized to zero. The resulting difference between the statistical and low-pass filter are shown in Figure 4.5 with a cutoff frequency of 0.503rad/s. Notice the phase lag, the statistical velocity jumps up and the low-pass velocity catches up to it over time. The same lag is seen when the statistical velocity drops to zero. However the low-pass filter smoothes out the sudden jumps which makes the follow vehicle's motion smoother.

4.1.4 Breadcrumb Uniqueness

The scenario with the lead vehicle remaining stationary creates problems because the visual sensors have noise. The noise appears to the path planner as if the lead vehicle is still moving. Therefore a desired speed is still calculated and the follow vehicle creeps toward the stationary lead vehicle. Additionally, if the target vehicle remains stationary, the breadcrumbs are added to the array and none are ever forgotten, which causes the breadcrumb array to run out memory. In order to combat these problems, breadcrumbs must be determined to be unique. The term "unique" is used to define whether a breadcrumb's position has changed significantly from the previous breadcrumb. In order to avert this problem, each new breadcrumb is compared to the last breadcrumb to see if it changed by a certain amount.

$$r = \sqrt{\left(x\left(t\right) - x\left(t - \Delta t\right)\right)^2 + \left(y\left(t\right) - y\left(t - \Delta t\right)\right)^2} \tag{4.8}$$

where x and y are the breadcrumb's positions at t the current time and $t - \Delta t$ the previous breadcrumb time, and r is a radius about the previous breadcrumb. A Tcl/Tk function is implemented that sets the minimum value for r during run time.

By setting a visual target stationary and recording the visual snake data, the standard deviation on the noise is found. Three times the standard deviation is what is currently being implemented in the path planner module which is a value of 5cm. If the breadcrumb is determined to not be unique, the desired velocity is set to zero and the module proceeds as if no breadcrumb data is received. Otherwise, the breadcrumb is added to the array of breadcrumbs and the processing of the breadcrumbs continues.

4.1.5 Initialization

In order for the least squares algorithm to obtain a useful cubic spline fit, there have to be four or more breadcrumb data points. When the first breadcrumb is received, a set of fake breadcrumbs are created. However, the desired velocity is not calculated until a second true breadcrumb is received from the visual snake sensor.

In order to get the best use out of the least squares fit, it is desired to have more than four breadcrumbs to fit to the cubic spline. When the first breadcrumb is received by the visual snakes, the first step is to define a line from the following vehicles initial position to the first breadcrumb. The line is used to attach artificial breadcrumbs. The question now becomes how many to add and where to place them on the line. How many to add is mainly dependent on your mission objectives but should only have a minor affect on the overall performance. In order to answer where to place the breadcrumbs, realize that not every breadcrumb can be saved to use in the least squares fit. Therefore, a parameter is defined called the forget distance and once a breadcrumb falls behind this distance from the following vehicle it is not required to remember it. Given the forget distance, it makes sense to add the artificial breadcrumbs to the line in equal distances between the new breadcrumb and the forget distance. A graphical representation of this idea is shown in Figure 4.6 where nine artificial breadcrumbs are added with a forget distance of 0.5m.

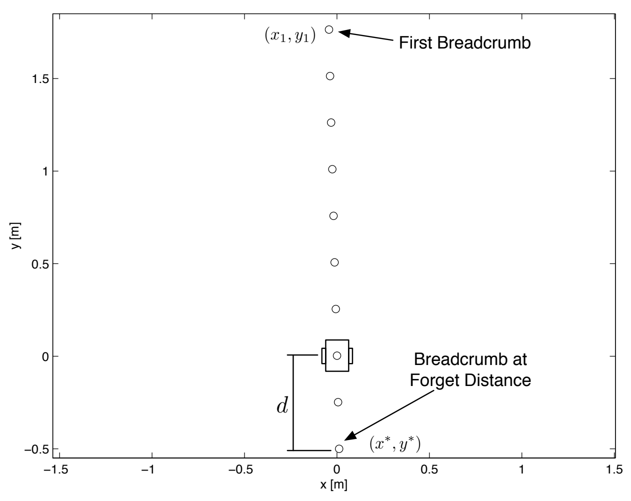


Figure 4.6: Setup for Initialization.

The slope of the line between the first breadcrumb and the following vehicle's initial position, which for the Pioneer is initialized to $(0, 0)$, is given as

$$m = \frac{y_1}{x_1} \tag{4.9}$$

where x_1 and y_1 correspond to the position of the first breadcrumb. The equation for any point along this line is then

$$y = mx \tag{4.10}$$

This is one constraint that must be met, the other is

$$d = \sqrt{x^{*2} + y^{*2}} \tag{4.11}$$

where d is the forget distance and x^* and y^* are position components at the forget distance. Solving Eq. (4.11) for y^* and substituting into Eq. (4.10), the value of x^* is found to be:

$$x^* = \pm\sqrt{\frac{d^2}{1 + m^2}} \tag{4.12}$$

The sign of x^* depends on the sign of the first breadcrumb's x value. Since the line crosses through the point $(0, 0)$, x^* has the opposite sign of the first breadcrumb's x value. This is checked by a simple if-then statement in the path planner module. The value of y^* is found simply by plugging x^* into Eq. (4.10).

The line is now divided up between the first breadcrumb, (x_1, y_1), and the forget distance breadcrumb, (x^*, y^*). The number of artificial breadcrumbs added is hard coded into the path planner module as the variable INITIAL_FILL which is currently set as ten. A time corresponding to each breadcrumb is required as well. Currently, the time at the forget distance breadcrumb is set to the negative of the time corresponding to the initial breadcrumb. Then the times are divided evenly among the remaining artificial breadcrumbs. These values fill the initial array of breadcrumb positions for the least squares algorithm.

Before leaving the initialization a few flags are set including the initialization flag so that initialization is not done every time. Most of these are flags for the code to run properly, however one important step is setting the desired velocity to zero. The first breadcrumb received gives no information about the speed of the lead vehicle. The artificial breadcrumbs are meaningless to calculate the speed because they are defined by the user and have no information about the lead vehicle's speed. At this point, the lead vehicle could be moving away quickly or at a dead stand still. So to be safe, the initial desired velocity is set to zero.

4.2 Propagating the Desired Position

When a new breadcrumb is received, the desired position can be calculated as previously shown. But what happens between breadcrumbs? A method is required to predict the desired follower position between visual sensor updates. Using the cubic spline path, the previous desired position on this path, and the desired velocity v_f, it is simple to propagate the prediction forward along the cubic spline. The estimated distance traveled is simply:

$$d = v_f \Delta t \tag{4.13}$$

Starting at the previous desired position, the cubic spline is once again divided into small intervals. The distances along each interval is added up until the estimated distance traveled, d, is exceeded. This position is the new desired position sent to the control module.

Note that because the filtered velocity is being used, the desired velocity changes each time the path planner module is run. Therefore, propagation must begin at the previous desired position and not the desired position found from the breadcrumb processing routine. Because the propagation starts at the previous desired position, errors in calculating the desired position accumulate over time. The cubic spline is therefore divided up into very small intervals, on the order of 10^{-6}, to avoid this error wind-up. Using intervals this small, the wind-up does not grow large before a new desired position is calculated from the breadcrumb processing routine.

The control module requires desired position and velocity, but it also needs to calculate the heading angle along the path. Therefore, the cubic spline is used to calculate this path angle. Taking the derivative of Eq. (4.2) and its $y(\tau)$ counterpart with respect to τ, the sensitivity of the cubic spline is found to be:

$$\frac{dx(\tau)}{d\tau} = a_1 + 2a_2\tau + 3a_3\tau^2 \tag{4.14a}$$

$$\frac{dy(\tau)}{d\tau} = b_1 + 2b_2\tau + 3b_3\tau^2 \tag{4.14b}$$

The sensitivities are then used to calculate the path angle as:

$$\theta_p = -\tan^{-1}\left(\frac{dx/d\tau}{dy/d\tau}\right) \tag{4.15}$$

4.3 UMBRA Path Planner Module

The UMBRA path planner module is shown in the high-level flow chart in Figure 2.1. The inputs and outputs to the path planner module are shown in Figure 4.7. The sensor data and path status flag are input from the interface module. The sensor data has units of meters. The Pioneer UGV outputs the inertial motion state sensed by the encoders in units of meters. The path planner solves for the desired position in meters, desired velocity in meters per second, and path angle in radians. These desired states are then output to be used by the control module. Determining the desired direction of the velocity and the desired heading rate is left to the control module. The control module is discussed next including the mechanics of the control and implementation.

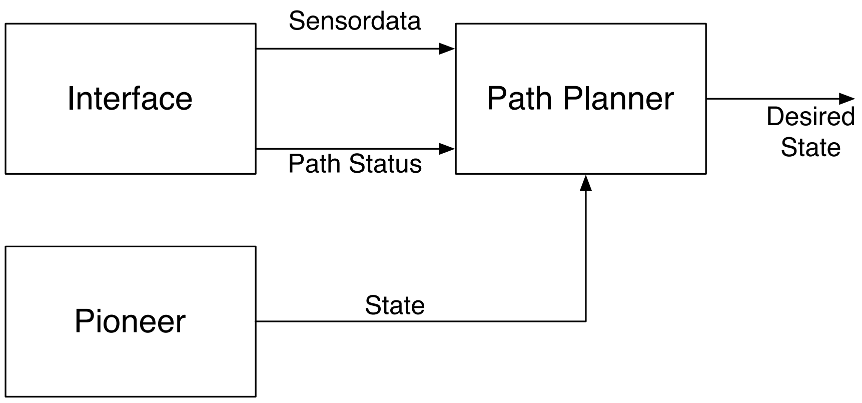


Figure 4.7: Inputs and Outputs to UMBRA Path Planner Module.

Chapter 5

Sliding Mode Control

A control law is needed to drive the follow vehicle onto the path described by the path planner module. A robust control law is needed because the kinematic equations are used to drive Pioneer. Pioneer itself is designed to be given wheel rates as control inputs. The required torques for each wheel, given the desired wheel rate, is then determined by the Pioneer's onboard computer. This may be thought of as another module that takes in the desired wheel rates and outputs the desired torques for the motors. However, this is all done by Pioneer's onboard computer. One benefit to using the kinematic model is that the same control law can be used on any size vehicle with different masses and inertias. A sliding mode control is chosen for this problem. The development and implementation of the control law will now be discussed.

5.1 Sliding Mode Control

Sliding mode control moves the system to a state where it is asymptotically stable. The goal then is to develop the law which moves the system to the asymptotically stable state in a finite time. In Reference 16 and 17, Slotine, Li, and Khalil develop a mathematically rigorous explanation of the sliding mode theory.

The kinematic equations of Pioneer are

$$\dot{\boldsymbol{X}} = \begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} -R_r \sin\theta & -R_l \sin\theta \\ R_r \cos\theta & R_l \cos\theta \\ R_r/L & -R_l/L \end{bmatrix} \begin{Bmatrix} \omega_r \\ \omega_l \end{Bmatrix} = [B]\, \boldsymbol{u} \tag{5.1}$$

where R_r and R_l are the radius of the right and left wheels respectively and L is the distance between the two wheels. The two control inputs ω_r and ω_l are the angular velocities of the right and left wheels respectively. The states x, y, and θ are the position and heading of Pioneer relative to the origin of an inertial frame (as discussed in Section 2.2.1). The inertial frame N is defined by the initial position and heading of Pioneer. The heading is defined as the angle taken counter clockwise from the initial forward direction of Pioneer.

A typical sliding surface as described in Reference 17 is

$$\boldsymbol{s} = \dot{\tilde{\boldsymbol{x}}} + [\Lambda]\, \tilde{\boldsymbol{x}} \tag{5.2}$$

where $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_d$ is the error in the state, $\boldsymbol{x}_d$ describes the desired path, and $[\Lambda]$ is a positive definite gain matrix. The time derivative of the sliding surface is therefore dependent on the dynamic equations $\ddot{\boldsymbol{x}}$. However, our control needs to be implemented on the kinematic equations. In order to account for this, the sliding surface is altered to the following form

$$\boldsymbol{s}(t) = \tilde{\boldsymbol{x}} + [\Lambda] \int_0^t \tilde{\boldsymbol{x}} dt \tag{5.3}$$

The time derivative of the sliding surface is now dependent on the kinematic equations $\dot{\boldsymbol{x}}$ as desired. When $\boldsymbol{s} = 0$ the system is in the sliding mode and is guaranteed to asymptotically converge to the desired state.

Lyapunov's direct method is now used to solve for control values that drive the sliding surface to zero. Choosing a quadratic Lyapunov function:

$$V = \frac{1}{2}\boldsymbol{s}^T\boldsymbol{s} \tag{5.4}$$

Taking the time derivative of the Lyapunov function gives

$$\begin{aligned} \dot{V} &= \boldsymbol{s}^T\dot{\boldsymbol{s}} = \boldsymbol{s}^T\left(\dot{\tilde{\boldsymbol{x}}} + [\Lambda]\,\tilde{\boldsymbol{x}}\right) \\ &= \boldsymbol{s}^T\left(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d + [\Lambda]\,\tilde{\boldsymbol{x}}\right) \\ &= \boldsymbol{s}^T\left([B]\,\boldsymbol{u} - \dot{\boldsymbol{x}}_d + [\Lambda]\,\tilde{\boldsymbol{x}}\right) \end{aligned} \tag{5.5}$$

Solving for the control input:

$$\boldsymbol{u}_{eq} = [B]^{\dagger}\left(\dot{\boldsymbol{x}}_d - [\Lambda]\,\tilde{\boldsymbol{x}}\right) \tag{5.6}$$

If the input matrix $[B]$ were square and invertible then $\dot{V} = 0$ which gives us a stable system. However, the pseudo-inverse, $[B]^{\dagger}$, is being used because the system has two control inputs to control three states. Multiplying a matrix by its inverse gives the identity matrix, but for the pseudo-inverse this is not true as seen

$$[B]\,[B]^{\dagger} = \begin{bmatrix} \sin^2(\theta) & -\sin(\theta)\cos(\theta) & 0 \\ -\sin(\theta)\cos(\theta) & \cos^2(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \neq [I]_{3\times 3} \tag{5.7}$$

It is never equal to identity for any value of the heading angle. At a heading angle $\theta = 0°$, the vehicle cannot be controlled in the inertial x-direction, and at $\theta = 90°$ it cannot be controlled in the inertial y-direction. This is because the vehicle is constrained by its wheels to only move in the body fixed y-direction. Pioneer must rotate 90° in order to move in the body fixed x-direction. Chris Romanelli's[7] control used two different descriptions of the heading angle to work around this problem. One heading angle was described along the path exactly as the path planner in this thesis does, and the other was described from the position of the Pioneer to the desired position. If the Pioneer needs to move in the inertial x-direction with a path heading angle of 0° it is not possible, but the angle to the desired point would be

90° and therefore controllable in the x-direction. However, this drastic heading angle would cause the visual target to move out of the camera image and relative position information is lost. The addition of a pan and tilt unit to allow the visual target to remain in view would alleviate this problem.

The control input is further modified to be driven towards the sliding mode by

$$\boldsymbol{u} = \boldsymbol{u}_{eq} - [B]^{\dagger}\,[K]\,\mathrm{sign}(\boldsymbol{s}) \tag{5.8}$$

where $[K]$ is a positive definite gain matrix. The current control law produces chatter due to the bang-bang technique of the sign function. A saturation function is instead used changing Eq. 5.8 to

$$\boldsymbol{u} = \begin{cases} \boldsymbol{u}_{eq} - [B]^{\dagger}\,[K]\,{}^{\boldsymbol{s}}\!/_{\varepsilon} & \text{if} \quad |\boldsymbol{s}| \leq \varepsilon \\ \boldsymbol{u}_{eq} - [B]^{\dagger}\,[K]\,\mathrm{sign}(\boldsymbol{s}) & \text{if} \quad |\boldsymbol{s}| > \varepsilon \end{cases} \tag{5.9}$$

This reduces chatter in the results. The system does not converge to the sliding mode as before but converges to an area surrounding the sliding mode defined by ε.

5.2 Implementation

Now that the control law has been developed, it needs to be implemented into an UMBRA module. The first thing to note is that the control input $\boldsymbol{u}$ is not the output of the UMBRA module. The Pioneer needs the desired forward speed and turning rate. The required torques on the two wheel motors to obtain the desired result is then calculated by the Pioneer's onboard computer. The control input, ω_r and ω_l, from the sliding mode control analysis are mapped to the forward speed and turning speed:

$$\boldsymbol{u}_{out} = \left\{ \begin{array}{c} \frac{R}{2}\left(\omega_r + \omega_l\right) \\ \frac{R}{2L}\left(\omega_l - \omega_r\right) \end{array} \right\} \tag{5.10}$$

where $R = R_r = R_l$.

Reviewing Eqs. (5.6) and (5.9), the state error, integrated state error, and the time derivative of the desired state are required to compute the control input. The state error is calculated as:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_d \tag{5.11}$$

where $\boldsymbol{x}$ is the vector of inertial position and heading fedback by the follow vehicle and $\boldsymbol{x}_d$ is a vector of the desired position and heading as given by the path planner module. The integrated state error is then simply:

$$\int_0^t \tilde{\boldsymbol{x}} dt = \int_0^{t-\Delta t} \tilde{\boldsymbol{x}} dt + \tilde{\boldsymbol{x}} \Delta t \tag{5.12}$$

This is a simple Riemann sum which locally estimates the integral as $\tilde{\boldsymbol{x}}\Delta t$ and adds it to the previous integral approximations. The derivatives are found in two ways, because the desired velocity and path heading is calculated in the path planner module and passed to the control module, the desired position derivatives are found by using these pieces of information as:

$$\begin{Bmatrix} \dot{x} \\ \dot{y} \end{Bmatrix} = \begin{Bmatrix} -v_d \sin\theta_p \\ v_d \cos\theta_p \end{Bmatrix} \tag{5.13}$$

where v_d is the desired velocity and θ_p is the path angle. The desired path angle rate is found by simply taking the derivative as:

$$\dot{\theta}_p(t) = \frac{\theta_p(t) - \theta_p(t-\Delta t)}{\Delta t} \tag{5.14}$$

However, as seen when finding the desired velocity in the path planner module, the derivative amplifies any small jumps in the path angle. In order to alleviate this effect, a low-pass filter is implemented again

$$\dot{\theta}_f(i) = \frac{1}{2+\Delta t\omega_c}\left[\dot{\theta}_f(i-1)\left(2-\Delta t\omega_c\right) + \Delta t\omega_c\left(\dot{\theta}_p(i) + \dot{\theta}_p(i-1)\right)\right] \tag{5.15}$$

where $\dot{\theta}_f$ is the filtered heading rate and ω_c is the cutoff frequency defined by the user. The initial value of $\dot{\theta}_f$ and $\dot{\theta}_p$ are set to zero. Since the path planner initializes by creating a straight line to the first breadcrumb, the initial heading rate should be zero.

5.3 UMBRA Control Module

The UMBRA control module is shown in the high-level flow chart in Figure 2.1. The inputs and outputs to the control module are shown in Figure 4.7. The desired state from the path planner is input to the control module. The desired state is the position in meters, velocity in meters per second, and path angle in radians. Encoders give the inertial motion state of the Pioneer in meters as an input. The control then uses the sliding mode control equations to find the wheel angular velocities. Equation (5.10) then maps these angular velocities to a forward speed in millimeters per second and turning rate in radians per second. All of the modules are now ready to be tested on hardware. The next chapter discusses two test runs and analyzes the data.

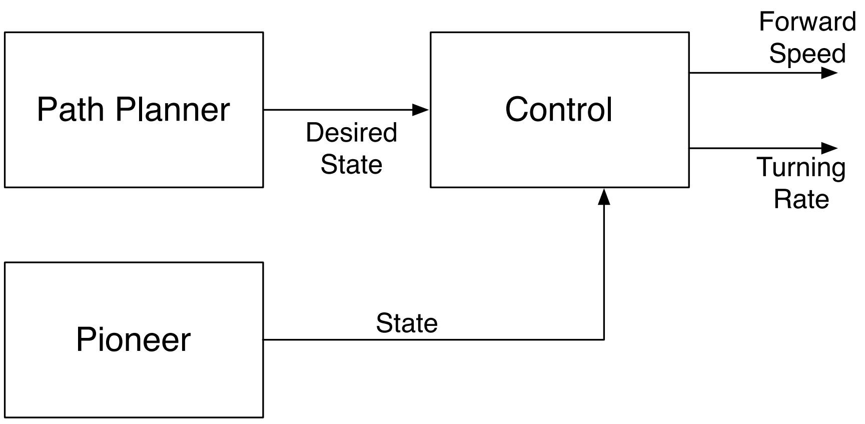


Figure 5.1: Inputs and Outputs to UMBRA Control Module.

Chapter 6

Results

Two test cases are presented. The first is a fixed visual target at 1.5m from the initial position of Pioneer. The following distance is set to 1.0m and the forget distance is set to 0.5m. Once Pioneer approaches the fixed target and becomes stationary at the one meter follow distance, the visual target is moved back 1.0m and held stationary again. In the second test case, Pioneer follows a visual target that is moving along a curved path. The target is initially 1.5m in front of Pioneer, and then follows along a circular path of radius 2.75m. The target is held parallel to the tangent of the circular path in order to simulate a lead vehicle moving along the curve.

In both test cases, the visual target is held by a human which results in some unsteady motions. The speed of the visual target along the curve varies due to this effect, as well as introducing yaw, pitch, and roll of the target.

Looking over the interface, path planner, and control modules, there are many variables that were described that must be given a value in order to run a test. These values are shown in Table 6.1 with the Tcl/Tk function call that is used to assign the values, the variable value, and a short description. The values remain the same for both tests.

Table 6.1: Parameters used for Hardware Visual Tracking Test.

Tcl/Tk function	Value	Description
snake setFrameRate	1 Hz	Sets the rate at which breadcrumbs are received in the visual snake module.
interface setDepthGain	50.067	Sets the depth gain in the interface module.
interface setFocalLength	835.235	Sets the focal length in the interface module
pathplanner setForgetDist	0.5 m	Set the forget distance used in the path planner module.
pathplanner setFollowDist	1.0 m	Set the follow distance used in the path planner module.
pathplanner setFilterFreq	0.2 Hz	Cutoff frequency, ω_c, used in the low-pass filter of the path planner module.
pathplanner setBCStdDev	0.05 m	Set the standard deviation used to determine uniqueness in the path planner module.
control setFilterFreq	0.2 Hz	Cutoff frequency, ω_c, used in the low-pass filter of the control module.
control Gains	“1 1 3” “0.1 0.1 3” 0.2	Set the gain values ($[\Lambda]$,$[K]$,ε).

6.1 Following a Fixed Visual Target

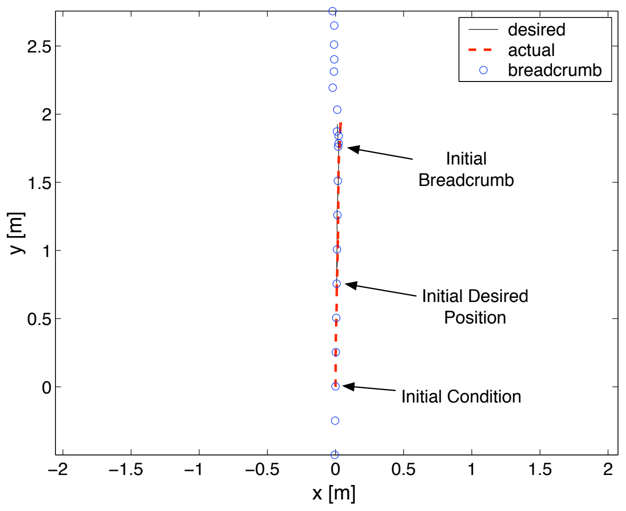
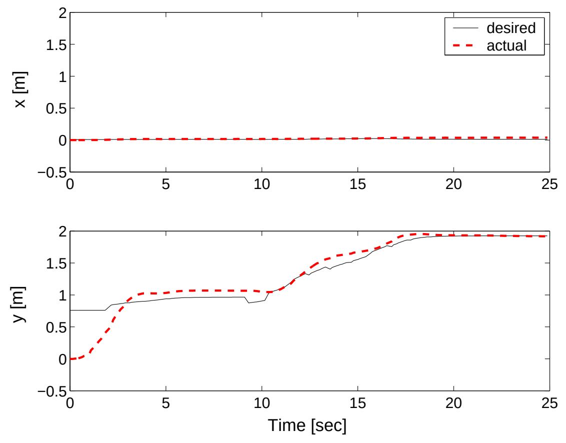


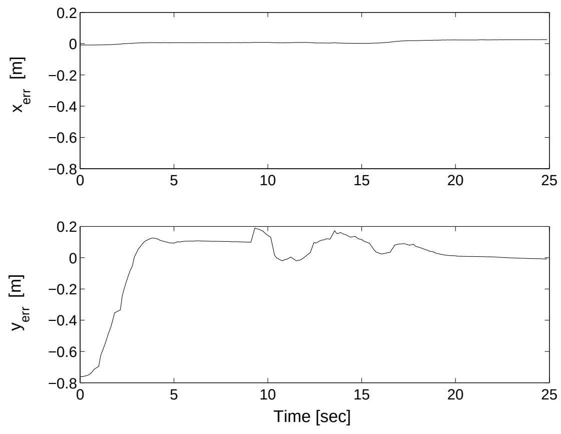
Figure 6.1: Inertial Position of the Desired and Actual Path with Breadcrumbs for a Fixed Visual Target.

A test is run where the visual target is kept stationary. Once the UGV reaches the desired follow distance, the visual target is moved 1m further away. The follow vehicle then reacts to this motion and again comes to rest at the follow distance.

The test is run successfully, and the position data collected is shown in Figures 6.1 and 6.2. The initial condition, initial desired position, and initial breadcrumb are marked in Figure 6.1. The breadcrumbs create a straight path at the beginning as desired by the initialization. The breadcrumbs then clump up at $(0.02, 2.0)$m where the visual target is being held stationary. However, the majority of the breadcrumbs are not processed because they are not considered to be unique. When a breadcrumb is not unique, the desired velocity is set to zero which is shown in Figure 6.3(a). When the visual target is moved further away, the desired path starts to move as well. The breadcrumbs appear about once every second during the motion which is exactly the update rate of the visual snakes. The motion stops and no more breadcrumbs are processed because of uniqueness. As expected, the desired



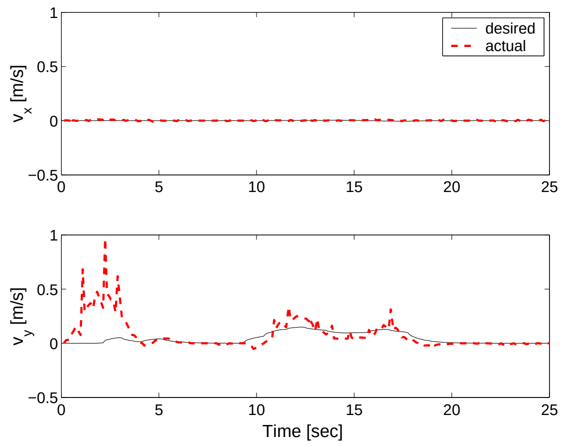
(a) Comparison of Actual and Desired Position



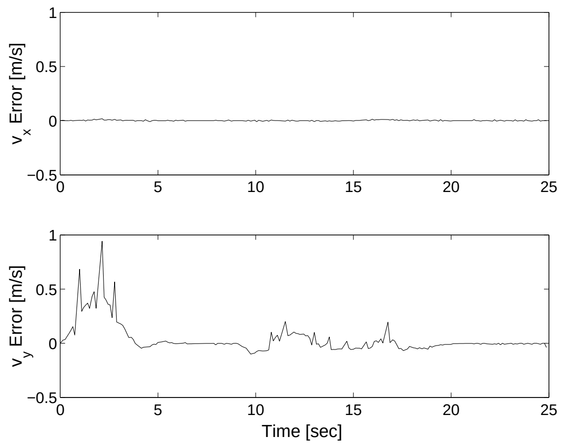
(b) Error between Actual and Desired Position

Figure 6.2: Inertial Positions for a Fixed Visual Target

path comes to rest. The desired path, however, has discontinuities in it. These are caused by new breadcrumbs and the forget distance because a new cubic spline is being fit. The spline will curve in a different shape and cause a small discontinuity. The hardware tests show that the control law handles this jump in data smoothly.



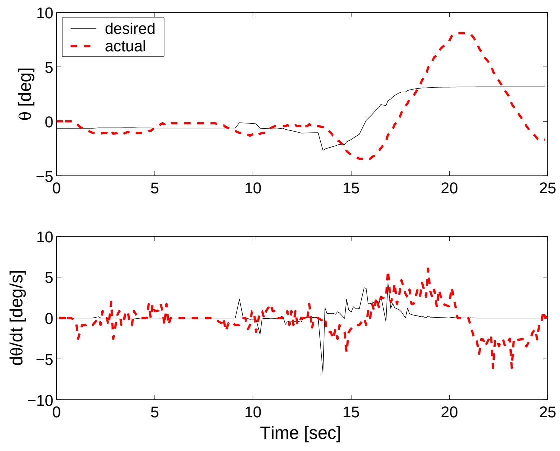
(a) Comparison of Desired and Actual Velocities



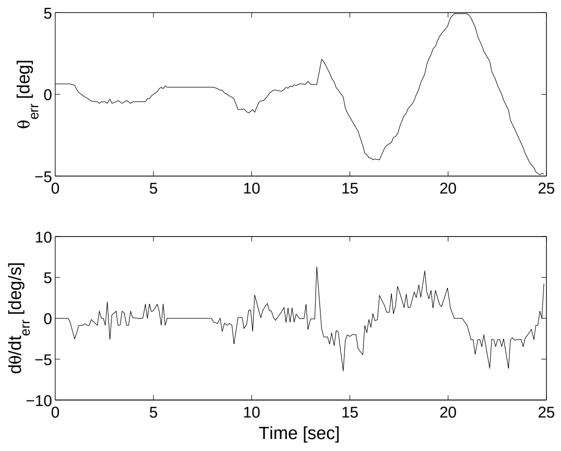
(b) Error between Actual and Desired Velocities

Figure 6.3: Velocity Data for a Fixed Visual Target

Ideally the path of Pioneer (shown as a dashed line) should overlap the desired path. The y-position follows the desired position closely and when the visual target finally comes to rest,



(a) Comparison of Desired and Actual



(b) Errors between Actual and Desired

Figure 6.4: Heading and Heading Rate Data for a Fixed Visual Target

it converges to the desired path; This is best seen in Figure 6.2(b) where the y-position error goes to zero. But, the error in the x-position grows by 2cm. However, note that precision station keeping is not required, and more importantly, the visual target needs to be tracked. The UGV's kinematics make it difficult for precise station keeping without losing sight of the target. As discussed in the chapter on the sliding mode control, two control inputs are being used to control three states and the x-position error is a result. The desired position is in the body fixed x-direction and to be controlled to this position, the heading of Pioneer must change. However, the desired heading is tangent to the desired path, so there is a competition over getting the x-position correct and keeping the heading angle correct.

The desired velocities are tracked well as seen in Figure 6.3. The actual velocity is much larger for the first five seconds though. The follow vehicle starts further away from the target than the follow distance. Therefore, the follow vehicle moves quickly to get to the desired position. This quick motion is seen as the initial error in the velocity.

The heading angle oscillates near the end of the run as seen in Figure 6.4. The oscillation is due to the competition between heading angle and x-position. As the y velocity comes

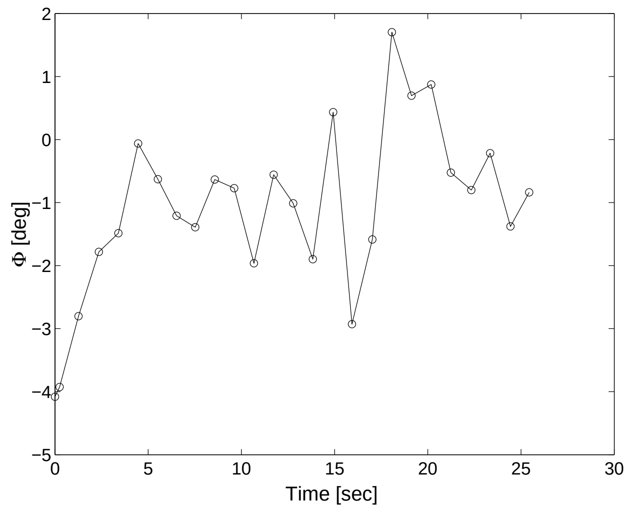
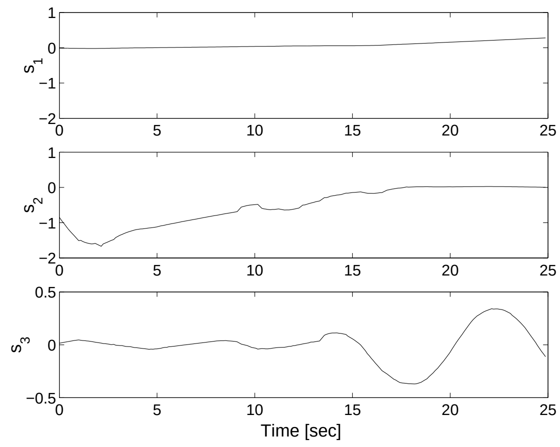


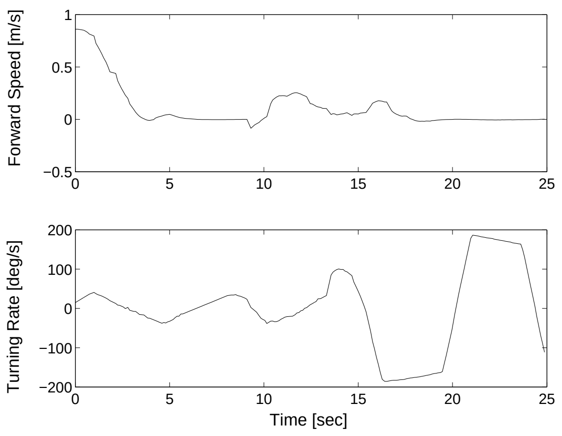
Figure 6.5: Principal Rotation Angle of the Visual Snakes for a Fixed Visual Target.

to zero a larger heading angle change is needed to correct for any error in the body fixed x-direction. The visual snake data shows more human error in the principal angle shown in Figure 6.5. The principal angle should remain near zero, but the majority occur around -1° with the largest being -4°. This is due to unintentional yaw, pitch, and roll of the visual target as it was held.

The sliding surface converges to the sliding mode for y but not for x or θ as seen in Figure 6.6(a). Because of the competition between the two states x and θ, the sliding surface never reaches zero as intended. Another reason for this could be the non-zero ε value used to remove chatter. However, the majority of the cause is due to the competing state variables. The control inputs give yet another view of the phenomenon in Figure 6.6(b). The turning rate oscillates near the end of the run as it attempts to correct for the x-position error. The forward speed control shows an initial burst of speed as the follow vehicle gets to the desired position. The forward speed command is zero where the target is being held stationary, and grows again as the target is moved further away. The follow vehicle catches up to the target and comes to a rest as expected.



(a) Sliding Surface Values



(b) Control Input Values

Figure 6.6: Sliding Surface and Control Input Values for a Fixed Visual Target

6.2 Following a Moving Visual Target on a Curved Path

A circle of radius 2.75m is marked on the ground. The visual target is moved by a human along the circle at a steady speed. The target is kept perpendicular to the tangent of the circle. The Pioneer starts 1.5m from the target and the target is parallel to the Pioneer's image plane. The objective is to track the circular path as close as possible.

The position data collected is shown in Figures 6.7 and 6.8. Additionally, the path that was setup for the visual target to follow is shown, although the person holding the visual target could not traverse the path exactly. Figure 6.9 shows the desired velocity which drops to zero corresponding to the spaces in the breadcrumbs. The control law is still able to move the vehicle to the desired path with minimal position error as seen in Figure 6.8(b). The position error remains under 4cm in both directions with no large spikes in the error. The velocity again spikes at the beginning as the follow vehicle catches up to the first desired position.

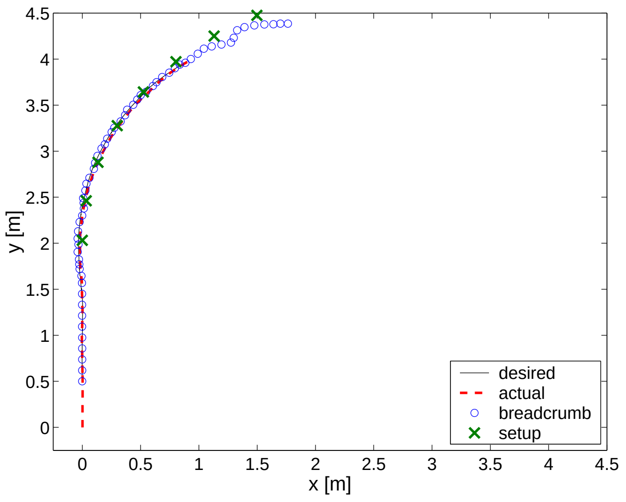
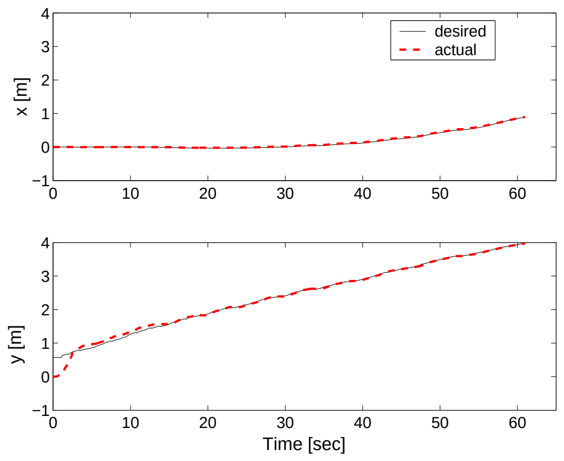


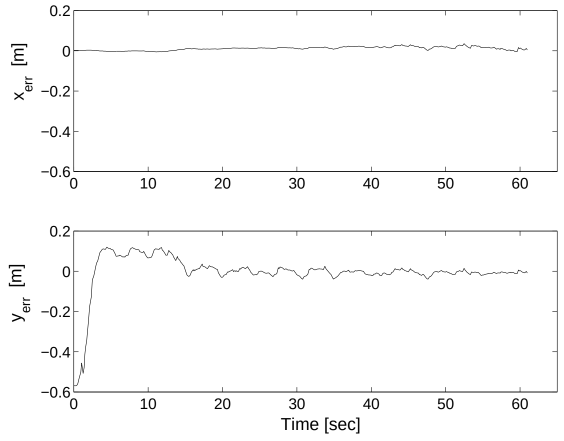
Figure 6.7: Inertial position of the Desired and Actual Path with Breadcrumbs for a Visual Target Moving Along a Curved Path.

The problem seen in the first test, dealing with the competing states in the control inputs, is hardly noticeable in this test. The reason is that Pioneer continues to move. The movement allows Pioneer to change its heading by a small amount to get large translations in the body fixed x direction. As opposed to the previous case where the follow vehicle was stationary and the heading angle would have to rotate 90° to fix any error in the body fixed x direction. The heading angle follows the desired heading angle well as seen in Figure 6.10(a). The oscillation about the desired heading is partially due to the gains trying to drive it to the path, but also where the control is attempting to fix any error in the body fixed x-direction. The error in the heading shows the oscillation better as seen in Figure 6.10(b).

The principal rotation angle switching phenomenon, discussed when formulating the interface module, is seen in this test. Figure 6.11 shows the principal rotation angle and few data points jump up near 90°. Yet the breadcrumbs received by the path planner are unaffected by the phenomenon because of the fix that was implemented. Ideally, the values should be either 0 or 90 if yaw is the only rotation present. Again, there is roll and pitch introduced



(a) Comparison of Actual and Desired Position

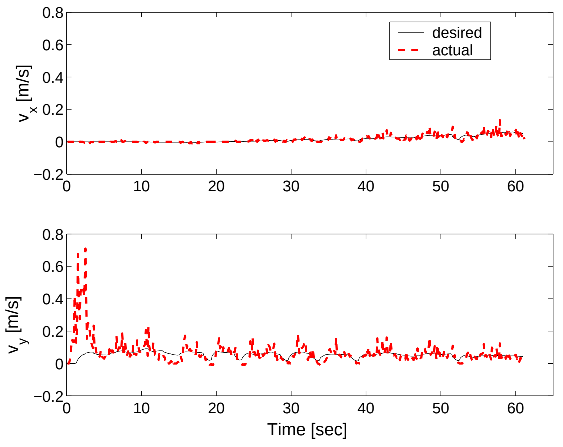


(b) Error between Actual and Desired Position

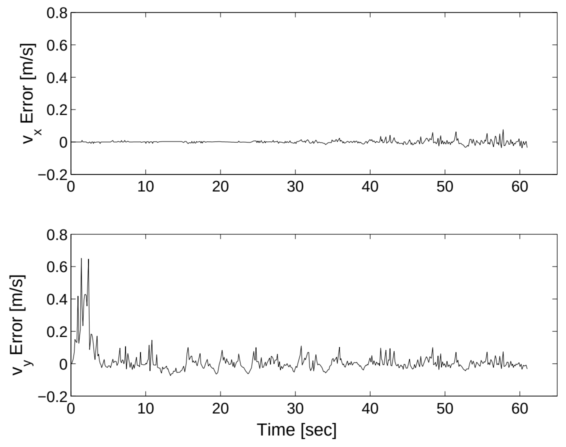
Figure 6.8: Inertial Positions for a Visual Target Moving Along a Curved Path

by human error as the visual target is moved. This shows robustness of the visual snakes and interface module.

As good as the results appear, none of the sliding surfaces converge to zero as desired, shown in Figure 6.12(a). Since the state errors seem to be small, the sliding surface is dominated by the integral term. Examining the state errors again, the position error seems to oscillate about a non-zero value. Because of the offset, the integration error builds up over time which is what is seen in the sliding surface. The reason is due to the three states being controlled by two inputs. The heading angle oscillates about zero, it does this because it is trying to fix the body fixed x-direction error, but at the same time it has to drive its own error to zero as well. The body fixed x-direction error is never driven to zero which accounts for the offset and the oscillations about the offset. The oscillation is also seen in the turning rate control input shown in Figure 6.12(b). The forward speed command jumps around a mean value of 5cm/s. The forward speed should be a constant value, however it is not because of the the same reasons the desired velocity is not constant. Two of these reasons are the target is not moving at a constant rate and the sensors have noise.



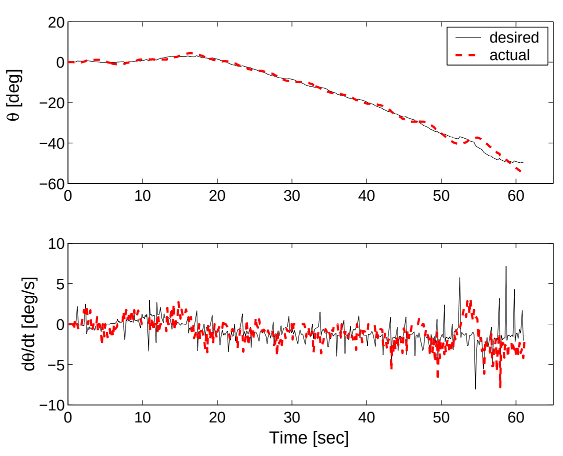
(a) Comparison of Desired and Actual Velocities



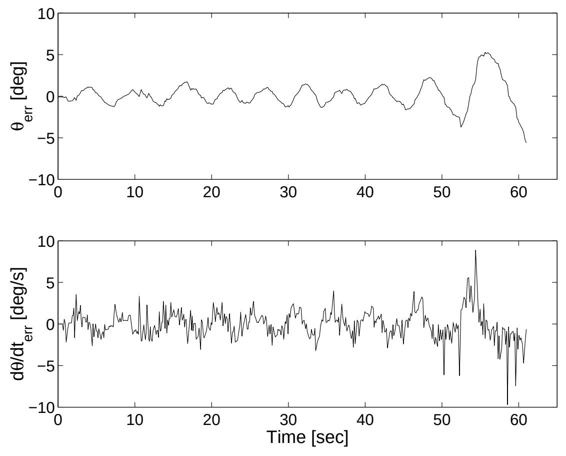
(b) Error between Actual and Desired Velocities

Figure 6.9: Velocity Data for a Visual Target Moving Along a Curved Path

Both tests show great robustness in the full solution. The interface module produces good information using the visual snakes. The path planner then uses these breadcrumbs to create a path that follows the breadcrumbs by a certain follow distance. The path planner's desired path matches up with the breadcrumbs. Finally, the control drives Pioneer onto the desired path with little error. The biggest problem arises from the control law because it uses a pseudo-inverse, but the results still seem robust to this effect.



(a) Comparison of Desired and Actual



(b) Errors between Actual and Desired

Figure 6.10: Heading and Heading Rate Data for a Visual Target Moving Along a Curved Path

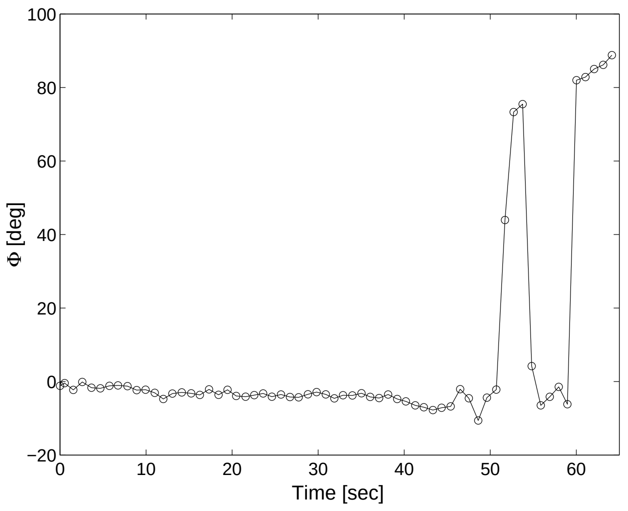
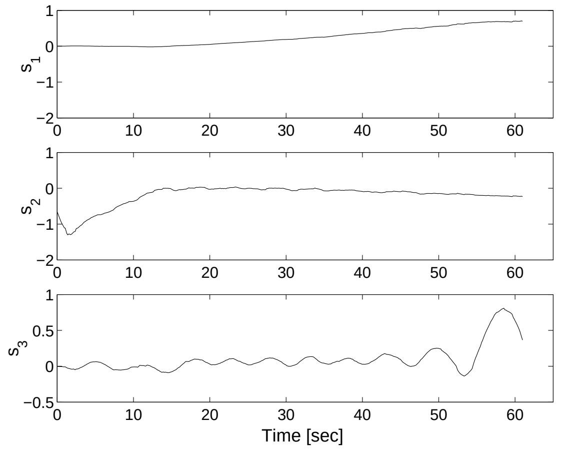
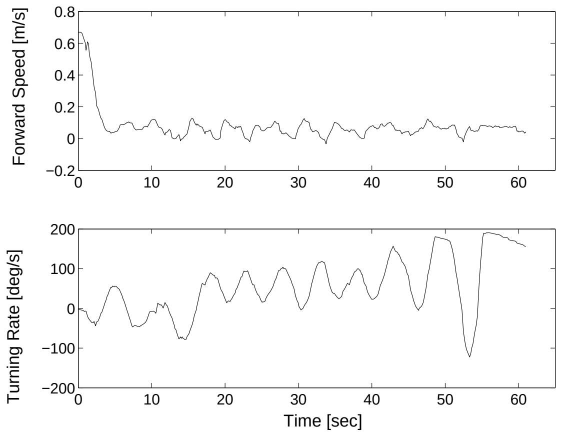


Figure 6.11: Principal Rotation Angle of the Visual Snakes for a Visual Target Moving Along a Curved Path.



(a) Sliding Surface Values



(b) Control Input Values

Figure 6.12: Sliding Surface and Control Input Values for a Visual Target Moving Along a Curved Path

Chapter 7

Conclusions

The problem of an Unmanned Ground Vehicle (UGV) following a human operated vehicle is investigated in this thesis. A possible solution is developed in the UMBRA simulation framework and tested with hardware. Statistical pressure snakes are used to gather the relative position information, and require no communication with the lead vehicle to obtain the information. The visual snakes are also shown to be robust to harsh lighting conditions. The path following solution developed allows the follow vehicle to avoid stationary obstacles, and provides for variable follow distances.

The three UMBRA modules are connected and run on actual hardware. Pioneer is able to be controlled on a fixed visual target as well as a target moving along a curved path. The combination of visual snakes and interface module produced robust results including situations where the switching method was required. The path planner is able to produce a desired path that followed the breadcrumbs. It created small discontinuities in the desired path because the cubic spline is updated every time a new breadcrumb is received. However, the control law works regardless of these discontinuities. Pioneer is controlled along the desired path smoothly and with position error under 4cm. The control experienced issues due to the three states being controlled by only two control inputs. It caused the heading to oscillate as it attempted to correct for any errors in the body fixed x-direction.

The visual path following algorithm developed in this thesis lays out a foundation onto which improvements can be made. A full stability analysis needs to be done on the sliding mode control. The hardware tests seem to show that it is stable, but it is evident that the pseudo-inverse creates problems. The fixed visual target test case showed this problem the best when the error in the x-direction did not converge to zero. Introducing deadbands on the error values might deter the competition between states. The topic of gain selection needs to be looked into as well. The two test cases used arbitrary gains; however, using a simple linear analysis to pick the gains might produce better results. Additionally, the linear analysis would give more insight on the how the gains affect the system by looking at decay time and damping ratio. The path planner works well, but still has areas that can be improved or at least explored for improvements. A pan and tilt unit could be controlled to keep the visual target centered in the camera view. This would improve sensor data at the edges of the camera frame as well as allowing Pioneer to follow along sharply curved paths. Finally, new sensors would like to be explored, such as LIDAR, to see the benefits and flaws of each. A combination of sensors will likely give the most accurate and robust results.

Bibliography

[1] M. Monda and H. Schaub, "Hardware Simulation of Relative Navigation using Visual Sensor Information," *AIAA Guidance, Navigation, and Control Conference*, Aug. 15-18 2005.

[2] "Virginia Tech - Space Systems Simulation Labratory," `http://www.sssl.aoe.vt.edu/`.

[3] "Free Flying Space Robots," `http://sun-valley.stanford.edu`.

[4] "NRL - Spacecraft Robotics Engineering and Controls Laboratory," `http://www.nrl.navy.mil/content.php?P=02REVIEW207`.

[5] "Mobile Robots - ActivMedia Robotics," `http://www.mobilerobots.com`.

[6] M. J. Monda, "Hardware Testbed for Relative Navigation of Unmanned Vehicles Using Visual Servoing," Master's thesis, Virginia Polytechnic Institute and State University, 2006.

[7] C. C. Romanelli, "Software Simulation of an Unmanned Vehicle Performing Relative Spacecraft Orbits," Master's thesis, Virginia Polytechnic Institute and State University, 2006.

[8] N. Kehtarnavaz, N. C. Griswold, and J. S. Lee, "Visual Control of an Autonomous Vehicle (BART) - The Vehicle-Following Problem," *IEEE Transactions on Vehicular Technology*, Vol. Vol. 40, Iss. 3, Aug. 1991.

[9] H. Schneiderman, M. Nashman, A. J. Wavering, and R. Lumia, "Vision-based Robotic Convoy Driving," *Machine Vision and Applications*, Vol. Vol. 8, No. 6, Nov. 1995.

[10] J. Gaspar, N. Winters, and J. Santos-Victor, "Vision-Based Navigation and Environmental Representations with an Omnidirectional Camera," *IEEE Transactions on Robotics and Automation*, Vol. Vol. 16, No. 6, Dec. 2000.

[11] A. K. Das, R. Fierro, V. Kumar, B. Southall, J. Spletzer, and C. J. Taylor, "Real-Time Vision-Based Control of a Nonholonomic Mobile Robot," *IEEE International Conference on Robotics & Automation*, May 21-26 2001.

[12] M. Inc., "Pioneer 3 Operations Manual," tech. rep., MobileRobots Inc., Amherst, NH, September 2004.

[13] H. Schaub and C. E. Smith, "Color Snakes for Dynamic Lighting Conditions on Mobile Manipulation Platforms," *IEEE/RJS International Conference on Intelligent Robots and Systems*, Oct. 2003.

[14] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, Vol. Vol. 1, Jan. 1988, pp. pgs. 321–331.

[15] D. P. Perrin and C. E. Smith, "Rethinking Classical Internal Forces for Active Contour Models," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. Vol. 2, 2001.

[16] H. K. Khalil, *Nonlinear Systems.* Upper Saddle River, NJ: Prentice-Hall, 1996.

[17] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

Vita

Charles M. Southward II was born in Alexandria, Virginia on April 11th, 1983. He grew up in the town of Warrenton, Virginia and graduated from Fauquier High School. Mike's college career began at Virginia Tech in the fall of 2001. He finished a Bachelor's Degree in Aerospace Engineering in the spring of 2005.