Summary of Computing Team's Activities—Fall 2007

Siddharth Gauba, Toni Ivanov, Edwin Lai, Gary Soedarsono, Tanya Gupta

1 OVERVIEW

The computing team accomplished the design and implementation of a sensor processing and navigation control system. This system receives inputs from the various installed sensors and provide an output to the motor controller in order to control direction. This system was customized in order to handle both the autonomous and navigation challenges of the IGVC competition. In sum, the following two diagrams sum up the high-level software design.

2 IMAGE PROCESSING

The camera was selected to function in outdoor environment under variable light conditions (bright or dim sunlight). A lens with wider view angle (81°) was chosen so that objects in the periphery, such as obstacles and lanes, can be registered for better navigation. A fish eye lens (107°) wasn't worthwhile since it doesn't work well in low light conditions and results in a larger distortion. Since two cameras were used for stereo vision the effective peripheral view included wider area. The camera selected was the Fire-I camera from Unibrain, a device tried and tested by many robots and competitors in the IGVC.

3 LINE AND OBSTACLE DETECTION

3.1 OVERVIEW

The goal of this section of the program was to facilitate a modular approach to line detection. The inputs are a color image, which comes from the digital camera. The output is a birds-eye-view binary map of the lanes, with which the navigation code will then process and couple with other sensory input to determine the best path. There are several steps involved in the image processing, which are outlined in the flow chart in Figure 3.1. The large parts include Inverse Perspective Mapping (IPM), Edge Detection, and Post Processing, as discussed in later sections.

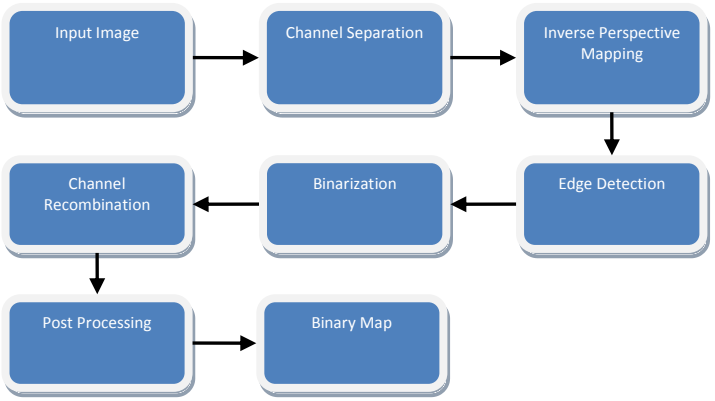

Figure 3.1: High level view of Image processing

3.2 CHANNEL SEPARATION AND EDGE DETECTION

The first step is to input an image, after which the channels are separated to achieve the appropriate color ratios for a given purpose. At the present time, this is done heuristically. For the competition, the lines are stated to be either yellow or white on top of green grass. For testing purposes, a sample image of white and yellow lines on a green field was used. The best color ratios found for white on green was 2*Blue – Green. For yellow it was Red – Green – Blue.

Edge detection finds lines where there is high contrast – an abrupt change from dark to light. There are several different edge detection techniques that can be applied such as Sobel, Prewitt, Roberts, Laplacian of Gaussian, zero-cross, and Canny. The best one was determined was the Canny method from a heuristic approach. There are several parameters that have to be set including the threshold for sensitivity as well as the standard deviation of the Gaussian Filter. In our case, edge detection can be applied to different color channels and then be recombined afterwards. The results of both channel separation and edge detection on white and yellow lines on green grass can be found in Figure 3.2.

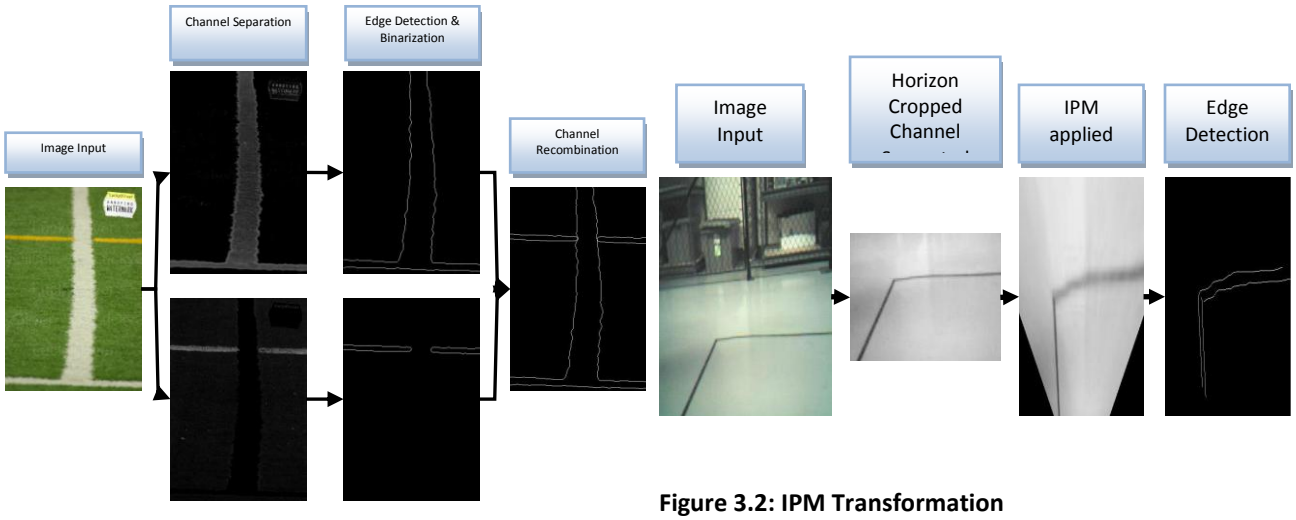

Figure 3.2: IPM Transformation

3.3 INVERSE PERSPECTIVE MAPPING

The two previous algorithms for channel separation and edge detection work perfectly fine with an image taken straight from a camera, however these images are not as helpful for navigation purposes because they are in a different perspective. This section discusses the technique of Inverse Perspective Mapping (IPM) to transform a view from a picture that is taken head-on into a birds-eye view as if taken from above. Essentially, it maps every pixel in a 2D perspective view of a 3D object and maps those pixels to a 2D planar view.

This transform is obviously dependent upon many factors, most importantly the aperture angle of the camera, the height of the camera above the floor, and the angle of attack of the camera vs the horizon. Both the aperture angle and the angle of attack determine the horizon below which the image is cropped. All three factors combined determine the skew of the IPM. These parameters must be recalibrated after installing a new camera or modifying the position of the camera on the robot.

In this example, the IPM transformation is done after the correct color channels have been acquired. Several images from the webcam were used to configure the IPM properly. Figure 3 & 4 show several examples of IPM at work on our test setup.

3.4 POST PROCESSING

After the recombination of the binary edges, there is an option for post processing. It would be nice to have a single solid line instead of the edges of a line. There are several techniques that can be used to achieve this goal, but the one that was chosen involves finding the distances for all pixels in the image from a non-zero pixel (in this case a line). That is, pixels that are very close to a line would have lower values, and pixels that are further away would have higher values. A threshold is then set and pixels below this threshold would be set to a binary 1. In other words, pixles that are close to the line are 1, and pixels further away from the line are 0. This in effect produces a solid bar surrounding any line. An example of this continues from the example in Figure 2 which can be found in Figure 3.

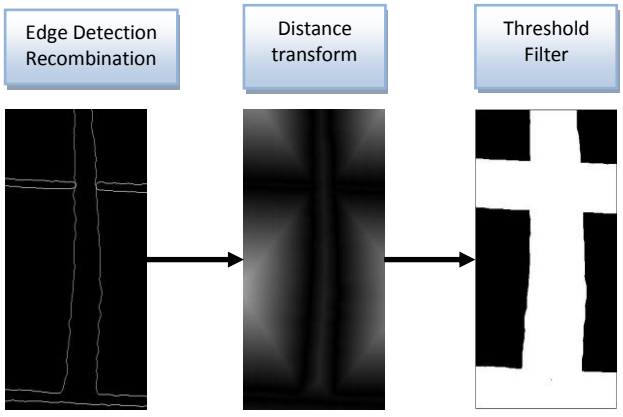

Figure 3.3: Post Processing

4 NAVIGATION ALGORITHM

4.1 WEDGEBUG ALGORITHM

The algorithm used by Rocky7 Mars Rover at the Jet Propulsion Laboratory offers a more cost efficient solution. The idea is to have the robot scan a sector of area centered in the direction of the goal for any obstacles. If there is an obstacle in the sector, the robot scans the sectors besides the previous sector until it finds a sector that has no obstacles. Once it finds a clear sector, it chooses this sector as the next direction to move. This is illustrated in Figure .

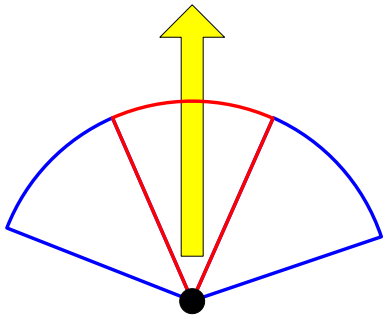

Figure 4.1: Sectors Checked by the Wedgebug Algorithm

The yellow line indicates the direction that the robot (the black circle) wants to go to. First, the robot checks if there are any obstacles in the red sector. If there are, it moves in the direction of the yellow arrow. Otherwise, the robot checks the blue sector for obstacles. If one of the sectors has no obstacles, the robot moves in the direction that the sector is centered at. Otherwise, the robot considers an even larger sector. This process repeats until a free sector is discovered.

To make the result more accurate, smaller sector angles are used. Since the length of the sector's arc can be less than the width of the robot, we look for multiple consecutive free sectors instead. The number of consecutive free sectors should be enough to provide the robot with a clearance to go through the sector.

This algorithm solves the computation problem faced by the D* algorithm and works for both the navigation and autonomous challenge. In the navigation challenge, the goal direction of the robot is obvious: it is the next checkpoint that the robot must go to. In the autonomous challenge, the goal direction is set to be straight ahead. This allows the robot to travel in a line as straight as possible, thus reaching the goal as fast as possible. However, the algorithm will perform poorly in U turns, as it will lead the robot to follow the outer curve of the U. Although this can be problematic if the outer curve is much longer than the inner curve, we will assume that such cases are rare.

4.2 DETERMINING GOAL SEQUENCE

IGVC's navigation challenge requires the competing robots to be able to visit around eight checkpoints in some order. To be able to finish the challenge as fast as possible, we must first determine the ordering of the goals to be visited. Since the obstacles in the course are unknown at the start, the sequence calculation is done while assuming that there is no obstacles in the course.

A possible algorithm to determine the shortest path is to calculate the total path for every permutation of goals and find the permutation with the minimum total path. While this may work with small number of goals, calculating for eight goals turns out to be computationally expensive (about 15 seconds). So, we use a nearest neighbour approximation with occasional adjustment.

The adjustment works as follows: When the robot is trying to go to goal A, it may move away from A when rerouting because of obstacles. When this happens, it checks if it is moving closer to some other goal B. If it is, and its distance is within a certain threshold, the robot switches its destination goal to B instead. The robot will resume moving towards A after reaching B.

4.3 DETERMINING POSITION AND ORIENTATION

We use different kinds of sensors to determine the orientation and position of the robot to make sure that the inaccuracy of one instrument does not affect the resulting calculation much. The orientation is defined to be the robot's bearing from the north direction, while the position is the (longitude, latitude) coordinate of the robot. To determine the robot's orientation, compass and IMU's yaw gyro reading are used. The IMU, along with the GPS and wheel velocity readings, are used to determine the robot's position.

To calculate the orientation, the IMU and the compass are used to calculate the robot's orientation. Calculating the orientation using the compass just involves reading the compass directly. To calculate using the IMU, it is assumed that the rotation rate of the robot since the last call to the update function is constant. Integrating the rotation

rate and adding it to the previous orientation gives the orientation of the robot.

The orientation of the robot is the weighted average of the two different orientations. To make sure that anomaly readings do not cause a large error in the calculation, an upper bound on the difference of the orientations is used. Let D_{IMU} be the orientation calculated using the IMU, and D_{Com} be orientation calculated using the compass. If D_{IMU} differs from D_{Com} by more than some threshold value T, D_{IMU} is set to D_{Com} + T.

Calculating the position of the robot uses the same steps as calculating the orientation. Difference in positions is calculated by calculating the Euclidean distance between two positions. However, since there are three values to consider, there is an additional step to take. Before adjusting the positions whose difference is greater than a threshold level, the algorithm determines if any of the three positions is an outlier. This is done by checking if any of them has differs from another by a value exceeding some threshold. If all three differences exceed the threshold, none of the positions is an outlier. If two of the differences exceed the threshold, the position common to both differences is considered the outlier and is removed. If one of the differences exceeds the threshold, the position that is further away from the position uninvolved in the difference is removed.

4.4 USING IMU TO CALCULATE ABSOLUTE POSITION

A problem with calculating the position using the IMU and wheel velocities is that we need to take into account the change in orientation through time. This is needed when calculating the robot's position using the IMU, since its reading is relative to the robot, while we need the absolute position. This means that if the robot continuously accelerates leftwards, it may not necessarily move leftwards. Instead, it can be traveling in a circle. To consider this, the orientation of the robot is assumed linear in time from the last call to this update function. So, the orientation can be expressed as a linear equation over time. Using transformation matrix, the acceleration of the robot through time in terms of (longitude, latitude) can be obtained. The position of the robot can be obtained by integrating this equation twice.

4.5 USING WHEEL VELOCITY TO CALCULATE ABSOLUTE POSITION

To calculate the position using the wheel velocities, we take the average velocities of the left and right wheels. The change in position of the robot can be calculated by using the formula in Equation 1, where x and y are the coordinates of the robot, w is the separation distance between the left and right wheels, v_R and v_L are the right and left wheel velocities respectively, and θ_0 is the initial orientation of the robot. After applying this equation, the new position of the robot is converted to (longitude, latitude).

$$x\ t = x_0 + \frac{w\ v_R + v_L}{2\ v_R - v_L} \sin\ v_R - v_L\ t/w + \theta_0 - \sin\ \theta_0$$

$$y\ t = y_0 - \frac{w\ v_R + v_L}{2\ v_R - v_L} \cos\ v_R - v_L\ t/w + \theta_0 - \cos\ \theta_0$$

Equation 1: Calculating Position Using Wheel Velocities[1]

4.6 DETERMINING PARAMETERS

The function uses six adjustable parameters, as outlined in Table 1. W_{IMU} is the weight for the position calculated using the IMU, W_{GPS} is the weighting for position calculated using the GPS, and W_{COM} is the weighting for orientation from compass reading. T_{POS} is the threshold for difference in positions, and T_{ANG} is the threshold for difference in orientations. OUTL is the threshold for determining outliers for the positions.

Name	Range Tested	Optimal Value
W_{IMU}	0:0.1:1	0.5
W_{GPS}	0:0.1:(1-W_{IMU})	0.1
W_{COM}	0:0.1:1	1
T_{POS}	0:W/5:W	0
T_{ANG}	0:1:5	4
OUTL	0:W/5:W	0.12

Table 1: Parameters to Determine Position and Orientation

For every test value of the parameter, we simulate the robot on a course, and check the orientation and position of the robot at the end of the course. The course is outlined in Table 2. The robot starts at location (0, 0) and oriented at 90°. At the end of every period, we calculate the readings of the GPS, IMU, compass, and wheel velocities by hand, and feed it to the update function after adding some errors. The errors used are outlined in Table 3.

To find the optimal parameters, after the experiment is run, we find a set of parameters that gives the most accurate readings for position and orientation. The optimal parameter is obtained by selecting the parameters that gives the most accurate position and orientation. The optimal parameters are outlined in Table 1, with 9.0 cm error in position, and 0.32° error of orientation.

Period (s)	Reading
0 - 1	Accelerate by 1 ms^{-2}
1 - 2	Constant velocity at 1 ms^{-1}
2 - 3	Turn right by 90° at 1ms^{-1}
3 - 4	Constant velocity at 1 ms^{-1}
4 - 5	Decelerate by 1ms^{-2}
5 - 6	Zero point rotation to the right by 180°
6 - 7	Accelerate by 1 ms^{-2}
7 - 8	Constant velocity at 1 ms^{-1}
8 - 9	Turn left by 90° at 1ms^{-1}
9 - 10	Constant velocity at 1 ms^{-1}
10 - 11	Decelerate by 1ms^{-2}
11 - 12	Zero point rotation to the left by 180°

Table 2: Course Used to Determine Parameters

Device	Error
IMU (Acceleration)	0.0196 ms^{-2}
IMU (Gyro)	100 / 3600 °s^{-1}
GPS	2 m
Compass	1°
Wheel velocity	0.2ms^{-1}

Table 3: Devices and the Errors Used in the Experiment

5 CONVERTING FROM GRID MAP TO RADIAL MAP

The map acquired from the image processing step is grid based. However, to make the map usable by the Wedgebug algorithm, it needs to be converted to radial coordinate map. This map contains the distance, in meters, to the nearest obstacle in a sector. The sector is specified by its angle from the direction that the robot is facing.

To do this, the algorithm assumes that the robot is at the bottom center of the grid map. To find the closest obstacle in a sector with direction θ, and sector angle of Δ, we sample the maximum distance travelable at angles equally spaced between θ and θ + Δ. For each sector, the minimum of such distances is taken to be the distance to the closest object in the sector.

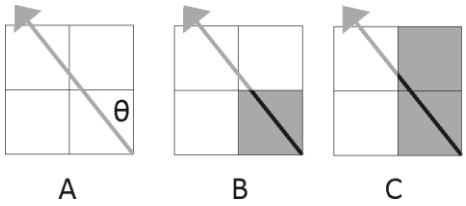

Figure 5.1: Determining the Maximum Distance Travelable

To find the maximum distance travelable at an angle θ, we draw a line from the position of the robot in the direction of θ, as illustrated in Figure (the robot is at the bottom right corner of the grid). First, check if the bottom right grid cell is an obstacle. If it is not, extend the line as shown in part B. The next grid cell to be checked is the top right cell. Again, if it is not an obstacle, the line is extended, as shown in C. This process continues until the cell checked is an obstacle or if the line has reached the edge of the map. Should the line reach the edge of the map, the maximum travelable distance is set to some value that is very large compared to the size of the map. This is done to make sure that the Wedgebug algorithm does not treat the edge of the map as an obstacle.

To make the process more efficient, when we are determining maximum travelable distance for angle θ in sector ϕ, the algorithm stops as soon as the length of line exceeds the maximum travelable distance for some other angle θ'.

6 FUTURE GOALS

Given the progress achieved this semester, the thrust in the next semester would be to prepare the robot for the IGVC 2008 competition. This is to be accomplished by sensor integration, testing, and timing optimization.

6.1 TESTING

A comprehensive testing plan for the robot is to be developed. Testing will be accomplished using resources such as the simulator and mock setups mimicking the competition environment. A monitoring console would be implemented

6.2 SENSOR AND HARDWARE INTEGRATION

All software will be migrated to a new, more powerful computing platform that meets the resource needs of the competition. New hardware such as the LIDAR, and sensors such as cameras, IMU, and SONAR are to be integrated into the platform to ensure optimal usage.

6.3 OPTIMIZATION

After successful algorithm validation via thorough testing, emphasis would be placed on code optimization to ensure rapid image and sensor analysis in order to ensure fast motion command generation. High algorithm throughput is required to ensure free-flowing robot operation.

Lucas, G.W. (2000). *A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators*/ Retrieved November 28, 2007, from http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html

Muad, A.M., Hussain, A., Samad, S.A., Mustaffa, M.M., Majlis, B.Y., "Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system", TENCON 2004. 2004 IEEE Region 10 Conference, Nov. 2004. Vol. A. pp. 207-210.

M. Bertozzi, A. Broggi, A. Fascioli, "Stereo Inverse Perspective Mapping: Theory and Applications", Image and Vision Computing Journal, 1998(16): 585--590, 1998.

E. Johnson, R. Hamburger, "CS5320/6320 Computer Vision Class Project", Weekly Report - University of Utah, March 12, 2007.