Localization and Tracking with a Wireless Sensor Network

Jonathan Schwarz
jonathan.schwarz@yale.edu
Davenport 2006
Electrical Engineering and Computer Science
Advisor: Andreas Savvides

I. INTRODUCTION

This project investigates using a sensor network to localize multiple robots on a grid, and to play a game with these robots. One possible game is to have both robots collaborating to collect falling targets on the grid as quickly as possible. Each robot is blind, that is, only using sensory data from the network itself.

II. GOALS

The purpose of this project is to explore what kind of game-like scenarios two robots that can communicate with a sensor network can play. Rather than have the robots do their own sensing and communication, they will be like thin-client robots, that is, all the heavy computation can be done on the network, not on the robot itself.

The robots play the game in the Becton CO-40 corridor, where a network of sensors has been constructed. Several of these nodes are outfitted with cameras in such a way that the entire hallway can be seen from multiple cameras. By using differently colored LEDs on each robot, it is possible to differentiate between the two and to localize them in the hallway on a known grid.

Once the robots can locate themselves on a grid by communicating with the sensor network, the next goal is to do something interesting with them. One such possibility is to have them collaborate in a task, such as dividing the responsibility of collecting scattered objects. The network will locate these objects and communicate their locations to both robots, which would work out amongst themselves which robot should collect each. This presents an interesting algorithmic challenge, and various heuristics could be tested against each other.

Another possibility is to have the robots compete. In such a game, one robot might attempt to hide, and the other robot would try to intercept it. In this scenario, both robots would be able to get the last known location of the other robot from the network.

Why a game? Games require more flexibility than other types of project, in which the tools can simply be tweaked as if for some kind of optimization problem. Good game design requires a strong combination of many attributes, leading to a more robust and scalable design. Games are also a good abstraction layer for some problems, letting real world details be removed or ignored to focus on the underlying technology. Finally, games are fun to work on, interesting to others, and easy to get involved with.

III. STATUS

The goals for the first semester of this project were to build the fundamental pieces needed to play the game. These are modules that can localize the robot and control navigation. As of the end of this semester, these modules are functionally complete. The robot can be localized within the field of view of a pair of cameras, and can navigate towards a fixed destination point. The navigation is still rough, but operable. The motor control is similar – it works, but needs more work before it is considered finished. The details of each module will be discussed in detail here.

IV. CAMERA MODULE

Several sensor nodes are equipped with a commercial, off-the-shelf (COTS) camera, similar to that of a cellphone camera. The camera nodes are capable of taking pictures of a scene as well as performing some basic image processing, such as Sobel edge detection, motion detection, and finding the direction of motion. The cameras have a resolution of 320x240 pixels, although smaller window comprising a subset of the image, typically only 64 pixels high, is often used to decrease processing time.

The camera nodes run a simple module called motion that applies a red filter to the RGB image, differentiates the image from the previous frame captured, and applies a threshold to all the pixels. This allows a blinking or moving red LED to show up as motion to the camera node. When using a 64 pixel high window of the camera, updated frames can be processed at 4.150 Hz, or one frame every 241 milliseconds.
V. MEASURED EPIPOLES

With two camera nodes that can view each other, finding distances to other nodes is possible. When a common node C is observed simultaneously by cameras A and B, the distances between A and C and B and C can be calculated. Normalized vectors based on the unit vectors from each camera are used to find the rotation of camera B from the perspective of camera A. This creates a linear system of equations that is easily solved to result in the length $l_{ab}$ and $l_{bc}$ in terms of lab. If $l_{ab}$ is known, all distances can be found by scaling them by $l_{ab}$. This technique is called measured epipoles, so called because the point of image plane that intersects the line joining the two cameras is known as an epipole.

![Diagram of measured epipoles in action.](image)

Measured epipoles has been implemented before, but only as Matlab code running on a PC. I ported this program to C code that would run on the nodes. This was more difficult than expected because of the lack of native matrix libraries in the standard C library for the Arm processor used. This module was tested and found to produce identical results to the version that runs on a PC.

VI. COORDINATE TRANSFORMATION

The end result of measured epipoles is a set of three-dimensional coordinates from the perspective of the first camera. As this camera is usually mounted high above the table or work space being used, and pointed downwards at an angle, this coordinate system is not practical or very user friendly. To work around this, a new system of coordinates was based, starting with the origin at the corner of the table used for the robot experimentation. The table is in the $x,y$ plane, and $z$ is vertical towards the ceiling. This system makes it easy to tell if coordinates that the robot is reporting are realistic at all, and also easy to measure values to check accuracy.

$$
\begin{align*}
\begin{bmatrix} x' \\ y' \\ z' \\
\end{bmatrix} &= \begin{bmatrix} a & b & c & tx \\ d & e & f & ty \\ g & h & i & tz \\
\end{bmatrix} \begin{bmatrix} x \\ y \\ z \\
\end{bmatrix} \\
\end{align*}
$$

To accomplish this, a set of transformation and translation values needs to be created. This can be accomplished by finding a set of four non-linear points and plugging the values into the equations that represent the matrices above, where $x'$ and $x$ and the other variables are actually a matrix of the four measured points in each of the two coordinate systems.

$$
\begin{align*}
\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \\
\end{bmatrix} &= a \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\
\end{bmatrix} + b \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\
\end{bmatrix} + c \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\
\end{bmatrix} + tx \begin{bmatrix} 1 \\
\end{bmatrix} \\
\end{align*}
$$

Once this is done, future translation of coordinates is simple by saving the values for $a$, $b$, $c$, $tx$, and so on, and by plugging in to the following equations.

$$
\begin{align*}
x' &= a_x x + b_x y + c_x z + tx \\
y' &= a_y x + b_y y + c_y z + ty \\
z' &= a_z x + b_z y + c_z z + tz \\
\end{align*}
$$

This way, the matrix only needs to be solved once, and coordinates can be transformed with similar linear equations.

VII. NAVIGATION

The navigation system works by calculating headings for the robot. Since the motion module that runs on the cameras sends updates through the measured epipoles module to the robot every time it detects motion, there is a constant stream of updated coordinates for the robot. Every time there is an updated position sent to the robot, the navigation module stores the position it received prior to the update. It constructs a vector that represents the current heading of the robot by using the difference in coordinates of the current and previous positions as components of the vector. The module also constructs a destination vector from the
current position to the destination, which is hard coded into the module at this stage.

The navigation module calculates the distance of the current location from the destination. If this is within a defined value, the robot halts. Otherwise, the angles of the heading and the direction to the destination are computed, using the arctangent of the y and x components of the vector. The angle is then corrected to the proper quadrant based on the sign of the components. Next, the angles are subtracted to determine how much of a turn the robot needs to be on the correct heading. Since the robot motor control only currently includes 90° and 45° turns, fine navigation is not possible. Instead, the unit circle is divided into rough sections, with any angle delta between -30° and 30° resulted in a forward command. If the angle difference is between 30° and 60°, the robot turns right 45°, and left 45° if it’s between -30° and -60°. More than 60°, and the robot makes a 90° right turn, and a 90° left turn if the angle is less than -60°. After each turn, the robot moves a small amount forward in order to trigger a position update with the robot on its new heading. This is a rough form of navigation, but it works. The robot tends to zigzag as it heads towards it destination because the response is essentially underdamped because of the lack of finer turns.

### VIII. ROBOT INTERFACE

The robot was originally intended to be controlled by two nodes, one a dedicated motor controller, and the other running localization and navigation modules. This module would send motor commands to the other node via a serial cable. However, there were issues in the hardware implementation of serial communication on the nodes, and it was decided to combine the two nodes into one in the name of expediency. Therefore, the motor control is simply another module, not a separate node. Currently, there are 9 commands. Each command is sent as two bytes, the first containing the command itself, and the second is a parameter. Only certain commands require parameters.

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Forward</td>
<td>Distance (cm)</td>
</tr>
<tr>
<td>1</td>
<td>Reverse</td>
<td>Distance (cm)</td>
</tr>
<tr>
<td>2</td>
<td>Right 45°</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Right 90°</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Left 45°</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Left 90°</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>180° turn</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Halt</td>
<td>Time (s)</td>
</tr>
<tr>
<td>8</td>
<td>Halt + clear queue</td>
<td></td>
</tr>
</tbody>
</table>

Table of commands

The motor module implements a queue of commands, up to a maximum of ten. When the queue reaches ten commands, new commands are ignored silently until slots are freed up.

There are several limitations to this kind of motor control. First, if commands are sent every time a position update is received, the robot can lag behind the commands that are sent in actually executing them. This leads to a robot that is always moving based on where it was several seconds ago, and will not navigate accurately. Secondly, the motor control is based on simple motor timings set by hand. The motors are not perfectly calibrated, and imperfections in the motors and how they respond, especially as battery voltages drop, result in unmatched motors. Straight lines and precise degree turns are impossible when this occurs.

### IX. FUTURE STEPS

The main upcoming work centers on actually playing a game with the robot. This semester’s work was all done with the goal of having a solid foundation for game development to take place on, and this has been accomplished. There are still some calibration issues to be worked out, but the modules all work as they should and interface well with each other without noticeable delay.

Despite the fact that the modules work well now, there are still some modifications to be made that would yield significant performance increases. Chief among
them is more advanced navigation. The ability to turn a precise angle, down to a degree or less, would greatly improve the speed and smoothness with which the robot navigates. Otherwise, all the precision with which the robot can calculate its position is simply thrown out. In addition, the motor timing can be improved by some calibration, possibly on the fly. Since the navigation module actually has accurate position data, and therefore speed information, it should be possible to adjust the motors to what their output is. In other words, if the robot turns two degrees left when told to go straight, the actual value sent to each motor can be adjusted based on how far off course the navigation module sensed it was after a forwards command.

Finally, most of next semester should be spent on game strategies and heuristics. Investigating interesting properties of robot collaboration and competition when their position info is known to themselves and the other robot should allow quite an interesting game to be developed.