MIDTERM REPORT

ELISE MCELLHINEY, ADAM VAN HAL

1. Progress

1.1. **Devices.**

- Imaged two identical Raspberry Pi 3's with Debian Jessie
- Installed OpenCV on Raspberry Pi 3's
- Established ways to clone implementations and snapshots between Raspberry Pi 3's
- Imaged the Intel UP board
- Installed ROS on the Intel UP board
- Software will be compatible with multiple devices so long as they have installed the dependencies of ROS and OpenCV

1.2. **Vision.**

- We ran a Raspberry Pi 3 with Debian Jessie and have established that OpenCV runs successfully with this setup.
- The Raspberry Pi 3 runs two simultaneous webcams in order to produce stereo images
- We have researched the capabilities of OpenCV running stereo calculations and decided that the depth map produced will be accurate enough to be usable for navigation.
- Each image is 640x480 pixels and thus utilizes approximately 2.5Mb of space. We have researched the capabilities of OpenCV running stereo calculations and have concluded that the depth map produced will be useful for navigation.
- The question of usability in this real-time application is more dependent on time than on quality of the images or depth-map. We are confidentOpenCV that we can achieve a usable depth-map utilizing two stereo webcams, however our initial timing data suggests that we may not achieve our current goal of a 200ms period using the Raspberry Pi 3.
- For the timings in the following table, we ran the image collection program with the two webcams a modest 10 times using the bash ”time” command to acquire some preliminary data. The process was run with no concurrent processes apart

Date: April 3, 2017.

from the OS itself.

$$\left[\begin{array}{cccc} & Average & Maximum & StandardDeviation \\ Real & 2.3535s & 2.452s & 0.0438s \\ User + System & 0.2344s & 0.42s & 0.1068s \end{array}\right]$$

- If we want updates every 10cm and to have the robot traveling at the maximum 0.5 m/s we will need to update every 200 ms. Obviously faster speeds will require a faster update period for the same result.
- As of now, the images are retrieved too slowly to be useful in navigation as you can see in the table above since the pair of images takes about two and a half seconds to run. We suspect that some of this is due to the fact that the cameras are reinitialized for each set of images. Implementation of ROS nodes to allow a single initialization and images to be produced periodically should improve the timings. We are currently unsure as to why we have such a disparity between the Real time and the User+System time since there could be a number of factors that slow down the Real time of the process.
- Debian Jessie also seems not to be easily compatible with the ROS installations that we have attempted. We installed Debian Jessie since we knew that it was compatible with OpenCV and looked to be compatible with ROS. This has proven not to be the case. We are currently in the process of installing the more dependable Ubuntu 16.04 Mate LTS and installing both ROS and OpenCV and are optimistic about the compatibility.
- We are also in the process of setting up the Intel UP board to produce depth maps.

1.3. Robot.

- Robot car is implemented with a Junior Runt Rover chassis and motors and is currently controlled by an Arduino Uno.
- The Arduino Uno can receive commands through serial inputs that translate into turning rates for the four wheels.
- Commands are configured to allow for forward, backward and turning motions on the robot car.
- More complicated commands can be configured for navigational purposes.
- With 2.5” diameter wheels running at about 140 rpm, the robot car can achieve about 1 mph (0.5 m/s) with the current pieces.

2. Plan

4-8-2017.

- ROS installed and running on both the Raspberry Pi 3 and the Intel UP.
- OpenCV running on the Raspberry Pi 3 with ROS nodes. This setup will allow co-development for the Pi and UP systems.
- Intel depth map camera working on the Intel UP

4-15-2017.

- Raspberry Pi 3 producing stereo depth maps using dual webcams
- Intel UP producing depthmaps
- Calibrate distance of depth map images
- Analysis of resolution of depth map images
- Timing analysis of production of depth maps
- Test preprocessing of images to reduce required bandwidth and improve results of disparity mapping

4-22-2017.

- Implement navigation system using depth maps for input
- Connect robot to navigation
- Have robot respond to navigation commands from controller
- Implement collision detection (Detect nearby surfaces and stop if one is too close)

4-29-2017.

- Implement "move towards furthest distance" navigation
- Improve navigation system and algorithms
- Utilize machine learning concepts such as neural networks for navigation
- Bug fixes

5-3-2017.

- Wrap up project
- Project presentation

3. Division of Labor

With the scope of the project as it is, we expect to both work on most components of the project. The listing below outlines parts of the project that we claim and will be responsible for fixing if unexpected problems occur.

Elise

- Implementation of physical robot
- Controls that translate serial commands into robot actions
- ROS module implementation
- Intel UP board and depth camera implementation
- Navigation based off of "best direction" data

Adam

- OpenCV interactions
- Raspberry Pi 3 and OpenCV processing of dual webcams into depthmaps
- ROS interactions with OpenCV
- Translate depth-maps into "best-direction" data