Ekin Gurgen, Michaela Hennebury

May 12, 2020

ELE302 Independent Project Report

Acknowledgements

We would like to thank Professor Jonathan Hodges and Professor Jeff Thompson for their continuous support and mentorship throughout the project as they provided us with the necessary tools and resources for developing Agromi. We would also like to thank David Radcliff and all the teaching assistants of "ELE 302: Robotic and Autonomous Systems Lab" for providing assistance that further enriched our understanding of electronics, feedback controllers and computer vision. Finally, we would like to thank Professor Robert Dondero for his excellent lectures in the course "COS 333: Advanced Programming Techniques" that assisted us in developing the user interface for Agromi.

1. MOTIVATION

Agriculture in the modern age is changing rapidly. Rising world population and labor shortages in farms necessitate more innovative solutions in the industry. These growing and changing demands can be supplied by utilizing robotic technologies to automate repetitive farming tasks such as planting, harvesting, pruning, mowing, and spraying. In this paper, we will introduce you the Project Agromi that aims to leverage on robotics and web development technologies to provide a small-scale, easy-to-use agricultural robot that can:

1) autonomously detect the dimensions of a rectangular field,
2) autonomously traverse the field in a desired "planting" path,
3) autonomously detect and avoid any obstacles on the path,
4) provide a clean and responsive user interface throughout the process.

2. BACKGROUND

2.1 Navigation of Mobile Robots

Navigation of mobile robots in environments with obstacles is one of the most fundamental problems in robotics. A classical approach to tackling the navigation problem involves solving three main subtasks: localization, mapping and motion planning. Localization is the problem of estimating the robot's pose relative to a given map of the environment. Methods such as dead reckoning or probabilistic localization algorithms that are variants of the basic Bayes filter can be used to perform mobile robot localization [1]. Mapping, on the other hand, is the problem of constructing a representation of the surrounding environment relative to the position of the robot. With the assumption that robot's pose is known, algorithms such as Occupancy Grid Mapping can be used to generate maps from sensor measurements [2]. When these two subtasks are performed concurrently, we arrive at the Simultaneous Localization and Mapping, commonly abbreviated as SLAM, which has become the standard solution to the mapping and localization problems in many robotic applications [3]. Motion planning is the task of determining a viable path from point A to point B in the configuration space. Obstacle avoidance is the core problem of motion planning and various discrete (e.g., Bellman-Ford, A*) and continuous (e.g., Rapidly-exploring Random Trees, Probabilistic Roadmap Method) path planning algorithms are commonly used to perform this task.

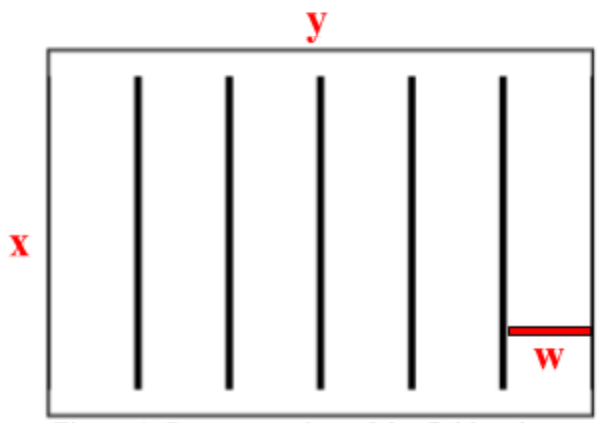

Figure 1: Representation of the field and planting lines given x, y, and w

In Project Agromi, we use a simple path planning algorithm to calculate a trajectory that traverses the rectangular field depending on the x, y, and w inputs from the user (Figure 1). We use dead reckoning (using encoder counts) to estimate the current pose of the robot relative to its initial position. And finally, we keep track of the locations of the detected obstacles relative to the robot's initial position and construct a simple feature-based map at the end.

2.2 Computer Vision

In the last decade, the use of computer vision for performing the navigation task for mobile robots has become a very powerful sensor modality. Vision can extract information beyond geometric representations of the environment and does not rely on having to emit signals. It provides an energy-efficient and light-weight sensor solution for the mobile robot navigation problem. In Project Agromi, we use the Pixy2 Camera and its relevant Python library to perform color detection.

3. DEVELOPMENT PROCESS

3.1 Building Agromi

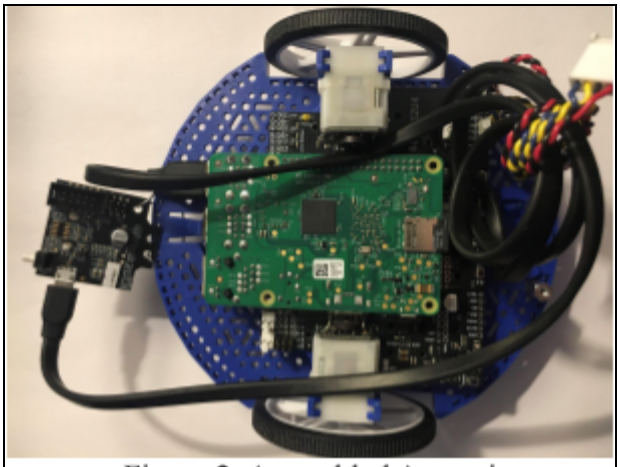

Figure 2: Assembled Agromi

The Agromi was built using a Pololu Romi chassis, a Romi 32U4 control board, Romi encoders, a Raspberry Pi, and a Pixy2 camera. Our robot was given to us with the hardware already built, so we did not need to assemble any components on that front. The Raspbian OS was used for the Pi, and we used a headless set-up (SSH from a laptop) in order to transfer files to and run programs on the Agromi.

The Pololu Raspberry Pi I2C Slave Arduino library was installed on the 32U4 control board, allowing the control board to receive I2C commands from the Raspberry Pi. These commands used in our Python code include setting the motor parameters and LEDs and reading

encoder values, button inputs and battery levels via the a_star library. This project exclusively used the motor setting and encoder reading capabilities, as discussed in the following section.

The Pixy2 camera was connected to the Pi via USB cable, and the pixy library was used to set the Pixy2 LEDs and interpret camera input. PixyMon software on a laptop was used to train the camera to recognize different color signatures. The camera was attached to the front of the chassis and was angled downwards so as to only take input from the floor directly below it.

3.2 Motion Commands and PI Control

Our initial Agromi program used fixed motor settings for the two Romi motors, but this quickly proved to be inadequate for the purposes of the project. First of all, the right and left motors behaved differently, requiring the two motors parameters to be set differently for the car to move in a straight line. Even if we managed to get the car to move correctly at one moment, however, changes in battery levels or floor surface would cause the car to turn and move at different speeds. Since the Agromi functionality requires it to navigate in straight lines and turn at precise right angles, another solution was required.

The robot includes "encoders," which use Hall sensor technology that allowed for speed control and dead reckoning positioning for the Agromi. The encoder readings were used for PI feedback speed control. PI control alone was still insufficient, however, because the two motors reached the target speed at different times. If one motor initially moved faster than the other, the angle of the chassis would change and cause the robot to progress in a different direction than was intended. To fix this, we added an angle correction subroutine. If the total encoder counts on the motors were more than 10 apart, a value of 40 was added to the parameter for the lagging motor. This significantly improved the problem, though the Agromi still had challenges moving in straight lines at long distances or after avoiding obstacles.

3.3 Pixy2 Camera and Obstacle Avoidance

The Pixy2 camera was used for corner detection and obstacle avoidance. Using the PixyMon software, the camera was trained to recognize red paper as signature 1 for obstacles and green paper as signature 2 for field corners (see Figure 3 below). Though we used red and

green for our project demonstration, it is very easy to reprogram the camera to recognize different colors.

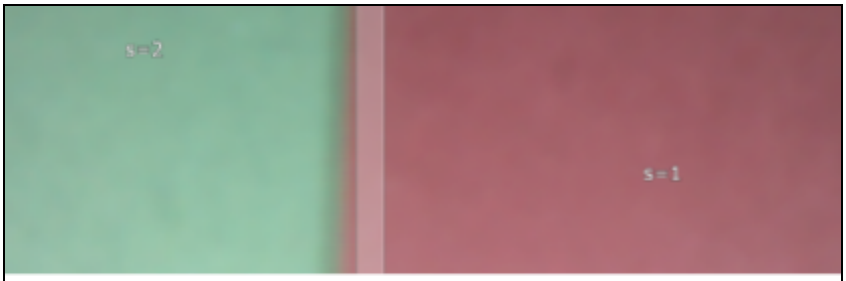

Figure 3: Training the Pixy Cam for Color Detection

After initialization in the Python code, the camera was set to the "color_connected_components" program which allows it to recognize blocks of color like those pictured. The LED lamp on the camera was also turned on so that the signature colors could be recognized regardless of the lighting in the room.

The objective of the corner detection method is to find the dimensions of a field for planting. Two corners of the field are marked with green paper. The robot moves forward until the camera detects the first marker, turns left, moves forward again until it encounters the second marker, and then returns to the starting position to begin planting. The total encoder count for the left wheel between the start and the first green marker is used to determine the y-length of the field, and the encoder count between the first and second markers is used to determine the x-width.

Obstacle detection occurs during the runPath method in Java. As the Agromi follows a path specified either by the user or by a previously executed field detection, it continuously checks for red obstacles. If it detects an obstacle, it enters a subroutine to avoid the obstacle, which is assumed to be 10cm long and less than 5cm wide. After avoiding an obstacle, the robot continues following the planting path.

3.4 User Interface

One of the primary objectives of the Agromi Project was to provide an intuitive and robust user interface for the farmers who would be using our robot. For this purpose, we wanted to create a one-page app that can take the necessary inputs from the user and show the progress of the robot

as it is running the planting job. Figure 4, given below, shows the technologies used in the development of the frontend and its communications with the backend.

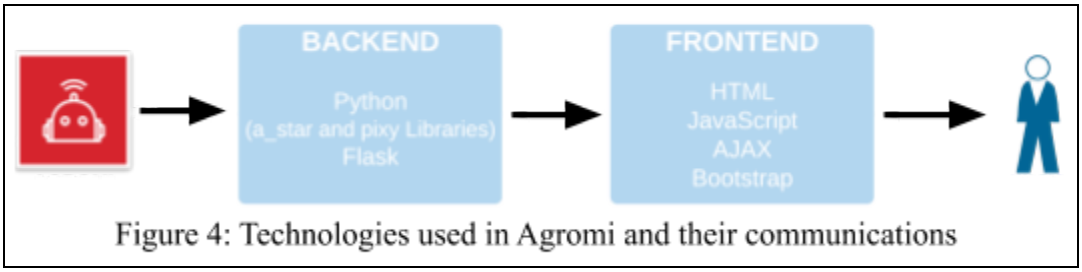

Figure 4: Technologies used in Agromi and their communications

When the client turns on the Agromi, a Python server starts running on their local network at port 5000. By typing the URL "http://*raspberry-pi-ip-address*:5000/" in their favorite web browser, where *raspberry-pi-ip-address* is the IP address of their Agromi's Raspberry Pi, users can access the "Agromi Control Panel". Note that Agromi Control Panel supports all browsers, including mobile browsers. Figure 5, given below, shows a screenshot of the UI.

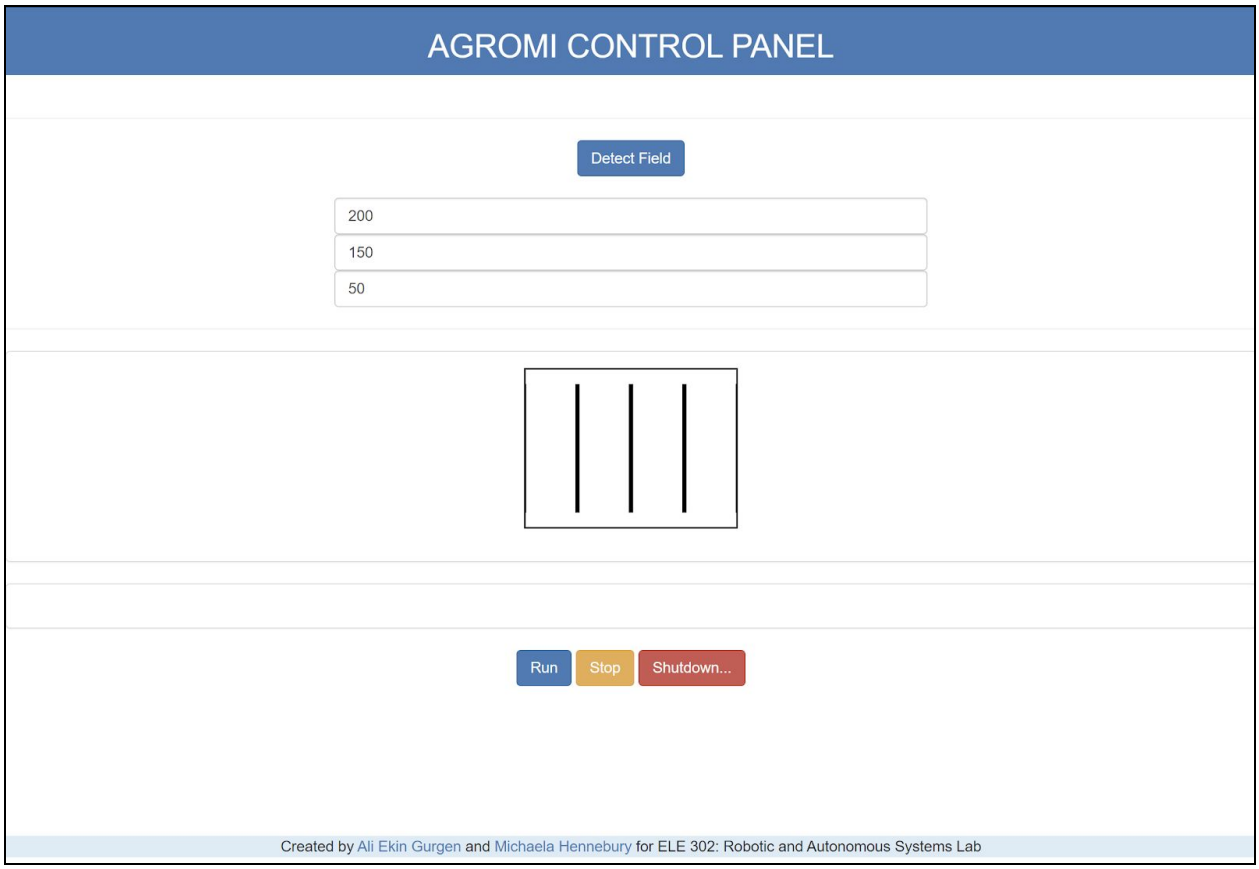

Figure 5: Screenshot of the Agromi Control Panel

We successfully achieved our frontend goals by creating a responsive one-page application that can communicate the location of the robot and any detected obstacles in real time between the frontend and the backend. With this web application, farmers can easily plan their planting job and monitor the progress as the Agromi is running the job.

4. CONCLUSION AND FUTURE WORK

There are many opportunities for continued work on Project Agromi, both on the smaller scale we have been working in and for future real-world implementations. As mentioned earlier, the robot often does not move in a straight line for long distances or after obstacles even with dead reckoning position detection. To fix this, a more robust position detection strategy must be used, such as GPS technology or a combination of other more advanced state estimation algorithms. Additionally, the current Agromi prototype lacks dispensing capabilities, which are critical for agricultural tasks like planting and watering. A future Agromi iteration would include dispensing mechanisms that could also be controlled via the user interface. Lastly, there are many kinds of obstacles that an agricultural robot might encounter in a field. Right now, the Agromi assumes that all obstacles are the same size and can be avoided in the same way, but a real-life model may need to react differently upon encountering a large mud puddle versus a cow. Additional work on the obstacle detection abilities of the Agromi could allow for avoidance of different sizes of obstacles and the potential to clear or remove certain types of obstacles.

There were four key goals for Project Agromi, all of which were accomplished in some magnitude in this work. The robot was able to detect dimensions of a marked field and detect and avoid obstacles using color-identification capabilities of a Pixy2 camera. It also autonomously traversed a planting path using PI speed control and dead reckoning positioning. Lastly, we were able to create an intuitive interface that provided the user with a live map of the environment, including field dimensions, obstacles and planting progress. While these tasks were accomplished for a small-scale, indoor robot, these concepts and capabilities are applicable in today's increasingly automated agricultural industry.

DOCUMENTATION

All program files can be found in the GitHub repository here:

https://github.com/aliekingurgen/romi-agri-robot

Files of interest include serverencode2.py and index.html (found in the templates folder).

REFERENCES

[1] Thrun, Sebastian, et al. “Mobile Robot Localization.” *Probabilistic Robotics*, 2000.

[2] Thrun, Sebastian, et al. “Occupancy Grid Mapping.” *Probabilistic Robotics*, 2000.

[3] Thrun, Sebastian, et al. “Simultaneous Localization and Mapping.” *Probabilistic Robotics*, 2000.