Agribot Technical Documentation

Georgia Institute of Technology

Andrey Kurenkov $andrey.sendme@gmail.com$
Troy O'Neal $tdoneal@gmail.com$
Pavel Komarov $pvlkmrv@gmail.com$
James Richard Nagendran $richardnagendran@bellsouth.net$

May 2, 2015
Rev 1.0

Contents

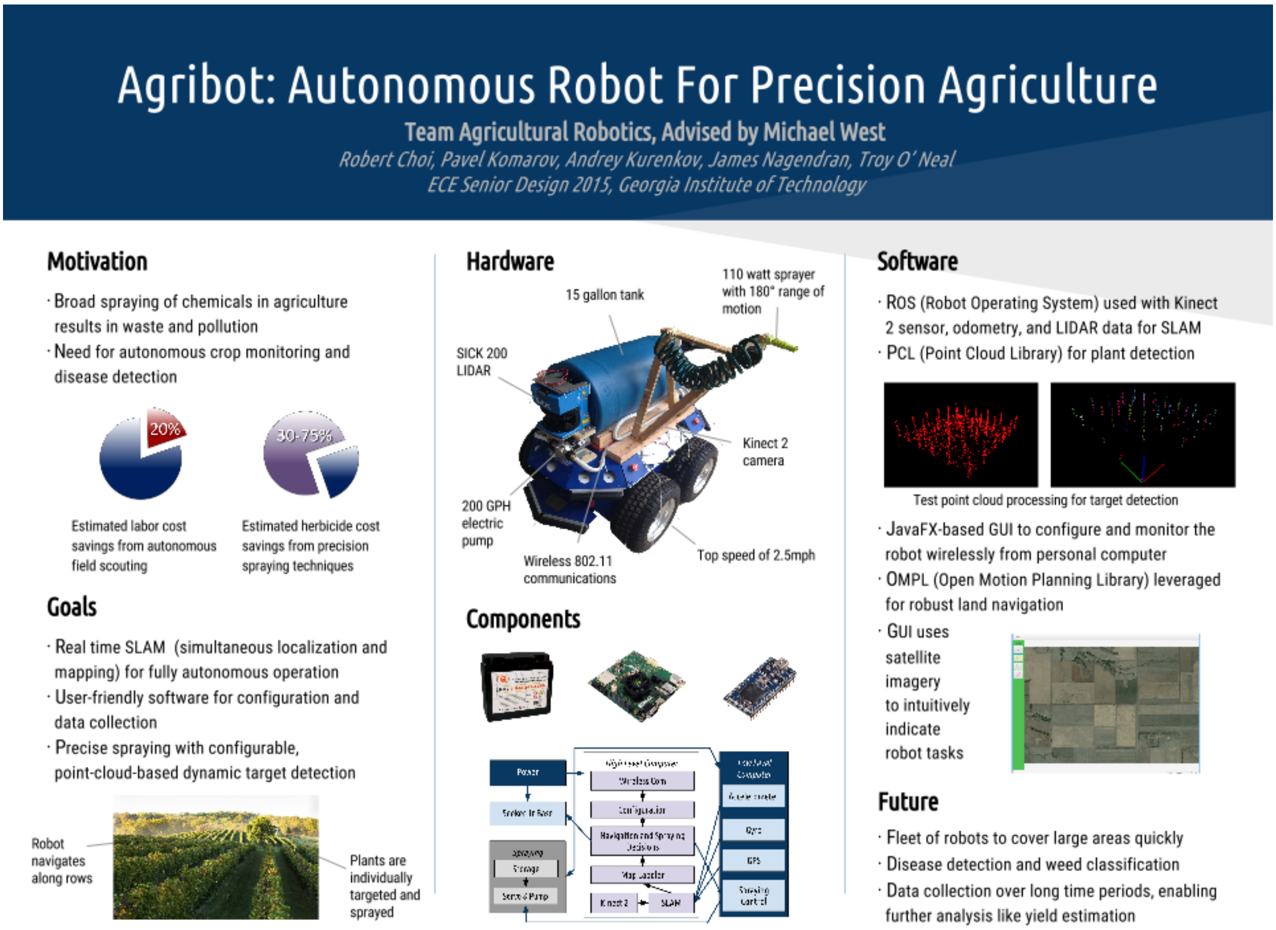


Figure 1: Poster overview of the Agribot project.

1 Overview

Agribot was a senior design project that aimed to develop a robotic platform that could autonomously spray plants on a farm while also localizing itself and making a high-fidelity map of the farm.

An overview of the motivation, goals, and final combined systems is presented in Figure 1. The rest of this document details the project to allow future teams to continue work from the point at which it has stopped.

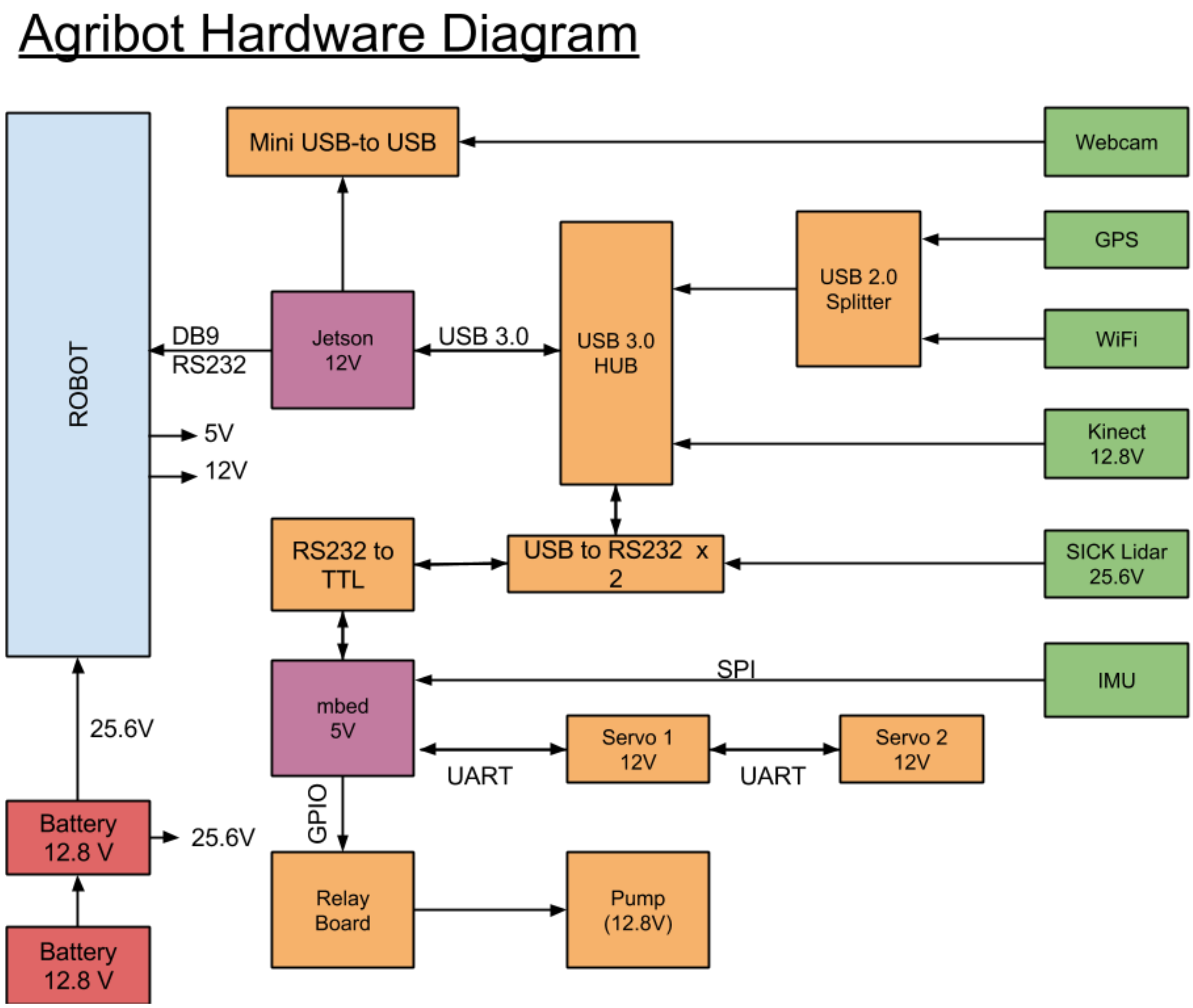


Figure 2: Hardware Diagram of Agribot.

2 System Hardware

2.1 Summary

As seen in Figure 2, besides the batteries and robot base, the overall system is made up of two computers (purple), multiple sensors (green), as well as various electronics (yellow). The sensors are necessary to perform localization and mapping, and the servos and pump are needed for spraying. The robot is a Seekur Jr research robot directly powered from two batteries, and it then outputs further voltages through a power board. These components are explained in greater detail in their respective sections.

Figure 3: The base Seekur Jr Robot.

2.2 Seekur Jr Base

The foundation of the system is the Seekur Jr research robot, commercially available from MobileRobots, Inc. This is a four-wheel differential drive robot which serves as the electrical and mechanical platform for all remaining systems. It can turn in place and its maximum speed is 2.5mph. With the current batteries, it should be able to operate for approximately three hours before recharging.

2.2.1 Batteries

The robot runs on an 18-36V power supply. In the current design, this power was pulled from a combination of two 12.8V 19.8Ah lithium iron phosphate batteries connected in series. The batteries are connected to a main power connector near the middle of the robot, distinguished by its orange and black wires. Ensure the polarity of the wires to this connector is correct, then screw the connector onto the robot connector until it locks in place.

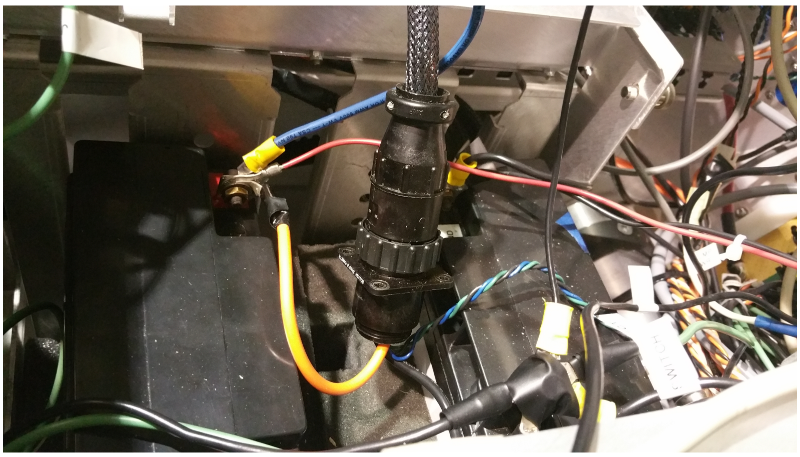


Figure 4: Batteries and main robot power connector

It is recommended to have a fuse inline with the batteries (30A-rated was used in the project), to mitigate the effects of any potentially catastrophic short-circuits. Once the power connector is secured, the robot may be powered on with the ON button at the back of the robot. Once powered on, the Seekur Jr provides +5V and +12V outputs from its internal PDB (power distribution board). These outputs are used to power the Jetson TK1 internal computer as well as the mbed embedded device. See the respective sections on these devices for more details.

2.2.2 Main Safety Switch and Fuse

A safety switch is inline with the 0V of the battery, and connects the battery ground to what is hereafter referred to as ”safety ground”. All loads are connected to safety ground as their (-) instead of directly to the batteries’ ground. When the switch is on, safety ground is shorted to battery ground, and thus current can flow normally through the batteries. When the switch is off, safety ground is disconnected from battery ground, and the path for current to flow is broken, disabling all electronics on the robot. A 30A automotive fuse is placed inline with the switch to avoid catastrophic short circuits, as these batteries are capable of driving hundreds of amps.

Figure 5: Main safety switch

2.2.3 Movement and joystick

The robot, once powered on, can be driven manually using the joystick. Plug the joystick into the connector near the ON button on the robot, and enable the motors. The MOTORS light should remain solid blue. Wait 45 seconds for the motors to fully initialize, then hold down the green GO button on the joystick. A green LED on the joystick should illuminate upon holding down the GO button. Once this is observed, the robot may be driven, using the joystick and speed dial to indicate the desired velocity.

The robot can also be driven from software, as this is the primary method of accomplishing autonomous navigation in this project. MobileRobots provides the ARIA library, which provides methods for setting the robots speed and orientation from a C++ class wrapper. Physically, the robot receives commands over an RS-232 serial connection. The RS-232 originates in the Jetson TK1 high-level computer and terminates in the robot at the following location. See the MobileRobots Seekur reference manual for more information.

Figure 6: Joystick active and ready for movement

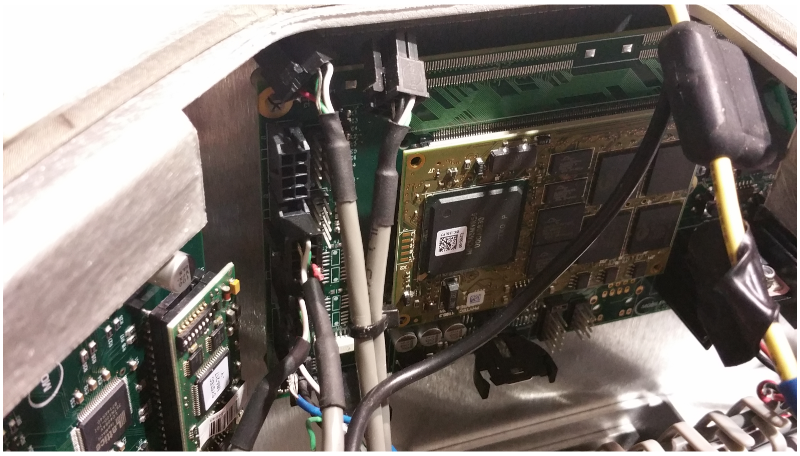


Figure 7: Location of RS-232 data connection to robot base

This connection is the only data connection made to the robotic base and is critical for sending motion commands and receiving odometry feedback from the wheel encoders. More details can be found in the Seekur Jr reference manual, available on request from MobileRobots.

2.3 Jetson TK1

The Jetson TK1 is a embedded computer available from NVIDIA. In this project, its purpose is to provide high level control logic for the robot. It contains 192 CUDA cores, which are utilized to perform SLAM, and it runs the navigation control loop to tell the robot where to go. It also communicates with the mbed to turn on the pump and sprayer servos when necessary. The Jetson community GUI should be consulted for more details about it.

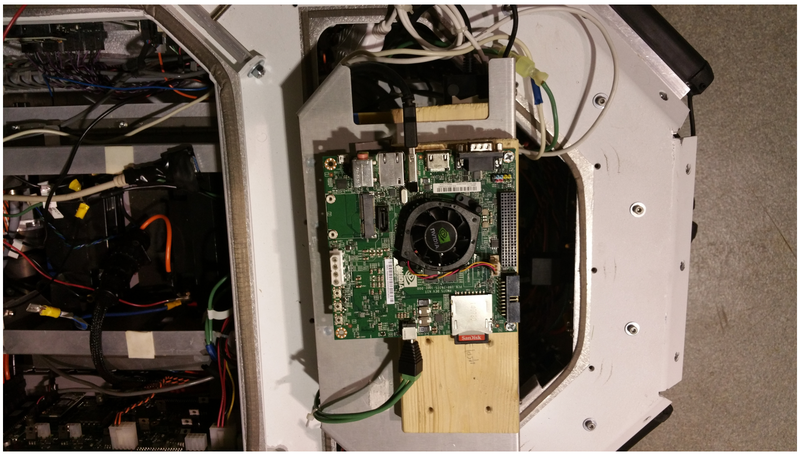


Figure 8: Jetson TK1 on mounting bracket

2.3.1 Development Access

As noted on the WiKi, the default username and password for the Jetson are both ”ubuntu”. These credentials are kept on the Jetson currently within the robot. When the computer is accessible outside the robot, it can simply be connected to an HDMI display as well as a mouse and keyboard, and used like any linuc computer. To access the Jetson when it is within a robot, a dlink router was used within the lab and the Jetson was made to connect to the ”dlink” network on power up. Using the router’s configuration address, the Jetson was then set to always receive the same IP address (192.168.0.155), which can then be used to ssh into the Jetson. In this manner, development can be done with the computer perpetually within the robot. In the future, it would be ideal to add connectors to the hardware within the robot to enable development on the Jetson with a display, since this is required to set it to connect to a particular WiFi network.

2.3.2 WiFi Hardware

To connect the Jetson to a wireless network it has a USB Plugable USB 2.0 Wireless N 802.11n 150 Mbps Nano WiFi Network Adapter. The GMYLE Wireless N/G 802.11n/g USB WiFi WLAN Network Adapter was also acquired and tested on the Jetson, with less success. As detailed on the Jetson wiki, the Grinch kernel was installed so as to make the Jetson have the needed drivers for these devices.

Unfortunately, the Jetson is unable to stay connected to GTwifi for more than a few minutes at a time, although it was able to stay connected to the dlink or other networks indefinitely. This problem remains to be resolved.

2.3.3 Wiring

The Jetson requires a power connection (dedicated wires come from the XJ2 PDB connector, these should be found and attached). It should also be attached via its standard USB port to the USB hub mounted on the top of the mounting bracket. An additional mini-USB to USB adapater may be attached if more USB ports are needed. The RS-232 wire from the

Seekur command and data port must be connected to one of the RS-232-USB adapaters on the Jetson, else the Seekur may not be communicated with. The Jetson requires at least 3 USB connections to function properly: two USB-RS232 connections to the Seekur and mbed respectively, and one wifi dongle to connect to a base station for monitoring and updates.

2.4 Mbed

The mbed embedded microcontroller board is used for low-level control and communication operations. For this project, it is used specifically to control the pump relay and to control the servos for aiming spray. Communication with the Jetson is performed over an RS-232 connection, with a level converter to lower the 12V Jetson data to 3.3V. The Jetson is mounted and wired on a breadboard, which has internal permanent wiring as well as several locations for connecting outside devices (the IMU and servos).

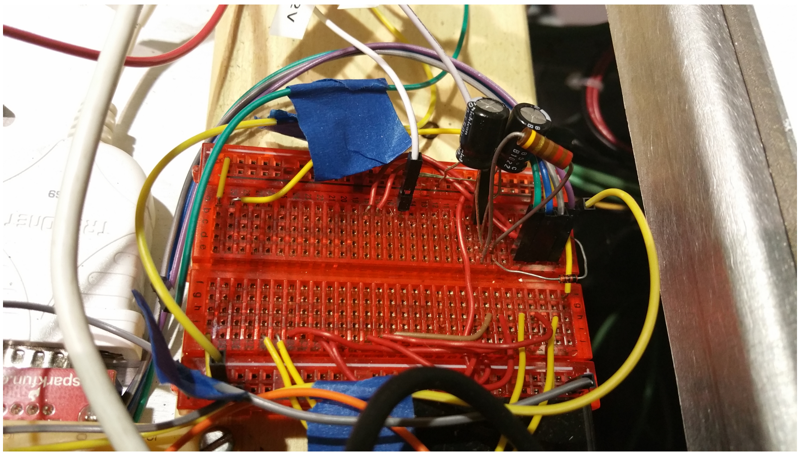


Figure 9: Breadboard where mbed goes - unfortunately the mbed itself was taken by a team member before the conclusion of the project

This processor was chosen at first to be a BeagleBone Black, but was later changed to the mbed LCP1768 due to flashing issues. Attached to it are the following modules: -2x Dynamixel AX-12A Servos (Section 2) -Silicon Sensing DMU02 (Section 3) -SparkFun RS232 Shifter (Section 4) Note that any of the parts above can be replaced so long as the code is rewritten to incorporate the new device. A compass was also planned to be included, but it was never implemented. Pump controls are issued here, and will be explained later. All details concerning the mbed are detailed in section 4.

2.4.1 Wiring

The mbed runs on the +5V and GND PDB screw terminals, and dedicated wires are provided as such. The appropriate +5V and GND pins on the mbed should be connected to these wires. The serial-TTL interface from the Jetson must be correctly connected to the pins of the mbed for the mbed to receive serial data. The breakout only has four header pins that are clearly labeled. Connect VCC(3.3V) and GND to the outputs of the mbed, and Tx/Rx to their respective pins on one of the Serial UARTS (Tx goes to Rx and vice-versa). An RS232 cable will attach to this and run to the corresponding socket on the Jetson. The mbed GPIO pin (documented in code) must be connected to the appropriate relay control wire (labeled ”mbed pump control”) for proper operation of the relay board.

2.5 Sensors

Multiple sensors are included on board the robot, and combined to perform robust SLAM as well as plant and obstacle detection.

2.5.1 Kinect V2

The Kinect V2 sensor is mounted on the side of the robot so as to face the plants while the robot drives by. It uses a standard USB 3.0 port to connect to the Jetson, although a new connector has been created to power the Kinect off of the battery power. An important note is that the box that connects to the barrel jack from the batteries appears to be broken following the connection of negative voltage to it - the Kinect has yet to be succesfully be powered on based solely on battery power, and this remains to be fixed.

2.5.2 SICK LMS 200

Lidar is mounted on the front of the robot to provide a way to sense objects and obstacles in the direction of motion. This hardware is powered directly by both batteries (24 V) and connected via serial-to-usb to the Jetson.

The SICK was a standard in robotics research for many years, so packages exist to interact with it in ROS. Utilizing these, we were able to read Lidar data. However, we did not complete integration of this data with SLAM, fully implement object-detection code, or complete the middle-ware to pass this data wirelessly to the GUI.

2.5.3 Wiring

The SICK LIDAR power cables should be connected directly between safety ground (-) and +25.6V from the battery (+). Marked connectors exist for this, as with other power wires. A grey RS232 with correctly soldiered pins has also been created, and should be used when connecting the SICK to the computer.

2.5.4 IMU

The IMU (also known as DMU) has eight pins that need connection to the mbed. The datasheet lists recommendations for connectors and crimpers, as the pins are too small and close together to use in a standard breadboard. Once the connections are in place, they go to the following pins on the mbed (as shown on the datasheet):

2.5.5 GPS

A Ublox M8 Evaluation Kit is connected to the Jetson via USB connection, which also provides power. It sends out data persistently to /dev/ttyACM0, and code has been written to parse these messages. This is documented under the Jetson software section of the document.

2.5.6 Camera

A Microsoft LifeCam HD 3000 was connected to the Jetson via USB and confirmed to work. As documented on the Jetson WiKi, multiple camera are compatible with the Jetson, and it is the intention of this project to use a camera to send visual data from the front of the robot back to the GUI. However, testing code for receiving data from the camera using OpenCV did not work, and so the camera was never integrated onto the robot.

2.6 Relay Board

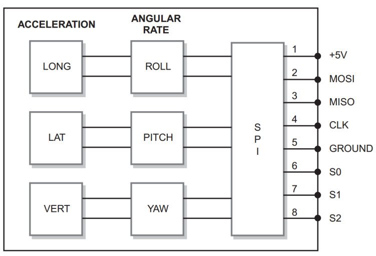
The relay board is a small protoboard mounted next to the mbed. It contains two relays, which cascade into each other to drive the high-power pump. The first (black) relay is very sensitive and requires a very low coil current, making it suitable for direct drive from the mbed GPIO. The second (blue) relay requires a bit more coil current but is easily capable of handling the 10A on its output contacts.

2.7 Servos

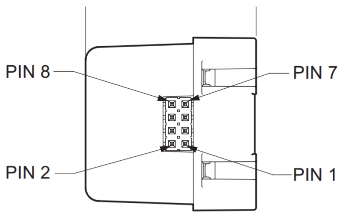
The robot's sprayer aiming apparatus is composed of two Dynamixel AX-12a servos attached to each other with plastic mounting brackets. An alternative choice to the AX-12A was the RX-24F, a similar and slightly more powerful servo made by the same company. This was replaced due to a complicated RS485 protocol, and a lack of time to get it working correctly. Instead, the AX-12A utilizes a simple Serial protocol. The communication format to the servo is the same for both.

2.7.1 Mechanical configuration

The servos are configured such that one rotates horizontally and one vertically, orthogonal to the direction of spray. In this way, we can aim the nozzle at any point in front of or below the sprayer-arm. The math to find the joint-angles to aim at a given location in space is implemented in comm.h within the /code/mbed/ directory.



(a) IMU data



(b) IMU pins

Figure 10: IMU schematics.

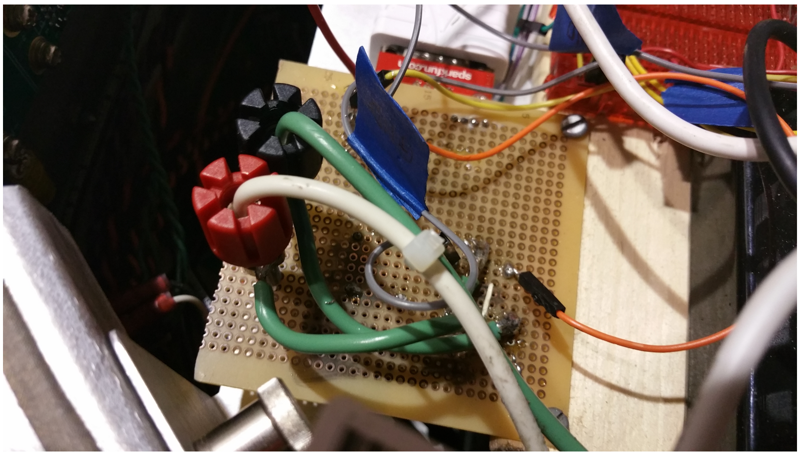


Figure 11: Relay board

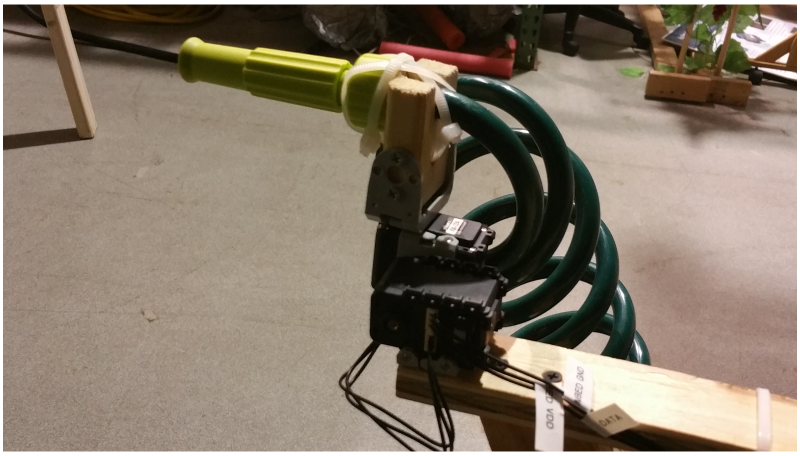


Figure 12: Servos and spraying apparatus

2.7.2 Data and Power wiring

Figure 5 contains the wiring diagram for the servos to connect to both the mbed and the power supply(in the final design, power comes from the robot). The DMU has eight pins that need connection to the mbed. The datasheet lists recommendations for connectors and crimpers, as the pins are too small and close together to use in a standard breadboard.

The AX-12A datasheet lists the ideal supply voltage at 11.1V, however it operates well at 12V with no noticeable change in function. The biggest note here is to use common ground between the power supply and mbed, which is why GND for the servo goes from power supply -¿ mbed -¿ servo to make it simple to wire. As far as Tx and Rx are concerned, any of the three UARTS can be used. They are connected to a single DATA line for the servo for read/write, which works due to the internal buffers written in the library.

To attach the second servo, simply plug it into the left slot of the first servo. Dynamixel servos each have an ID that you can program such that any command that is sent out will reach every servo in the chain, but only be accepted by ones with the corresponding number.

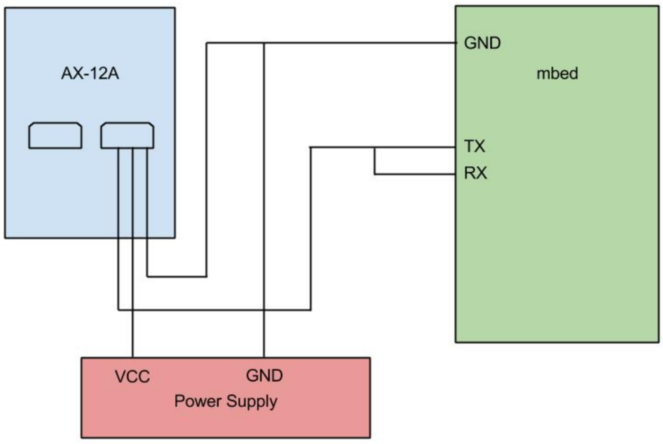


Figure 13: Servo to mbed wiring

2.8 Pump

The pump is a Pacific Hydrostar 12 Volt Marine Utility Pump. It pumps at a rate of 260 gallons per hour and at a pressure of 50 psi. Basically, it's super-powerful. Warning: do not run the pump dry. Ensure that the tank water level is high enough such that liquid may flow through the hose before attempting to activate the pump.

2.8.1 Hoses

The hoses are common garden hoses, which fit to the tank, the pump, and the nozzle. Because the end of the hose must be flexible to allow proper movement of the servos, the last few feet are a bendable spiral hose. It is imperative that the hoses be monitored diligently for leaks, as water leaking inside the robot could be catastrophic.

2.8.2 Wiring

The pump (-) should be connected directly to safety ground through its dedicated, labeled wire. The pump (+) should be connected to the relay board through the indicated connector.

2.9 Mounting Bracket

The mounting bracket is a removable metal plate that fits securely in the interior of the robot near the back. The bracket contains wooden mounting panels on which power terminals, the relay board, the mbed breadboard, the USB hub and the Jetson are mounted. The Jetson is mounted on the back side of the board. The mounting bracket may be inserted into and removed from the robot by pullable latch buttons located on the bracket.

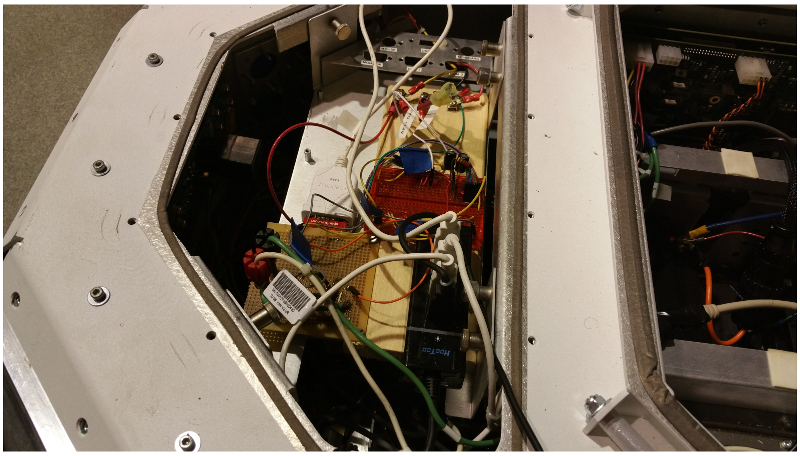


Figure 14: Mounting bracket shown inside the robot

2.10 PDB connections and power terminals

The +5V and +12V connections for the mbed, Jetson, and servos, and intermediate relay are powered from the robot's PDB (power distribution board). Note: the pump is not powered from the PDB, as it requires around 10A. It receives its power directly from Battery 1. The Jetson is powered from the XJ2 connector on the PDB and the remaining devices that run on +5V and +12V are powered from the WJ2 connector.

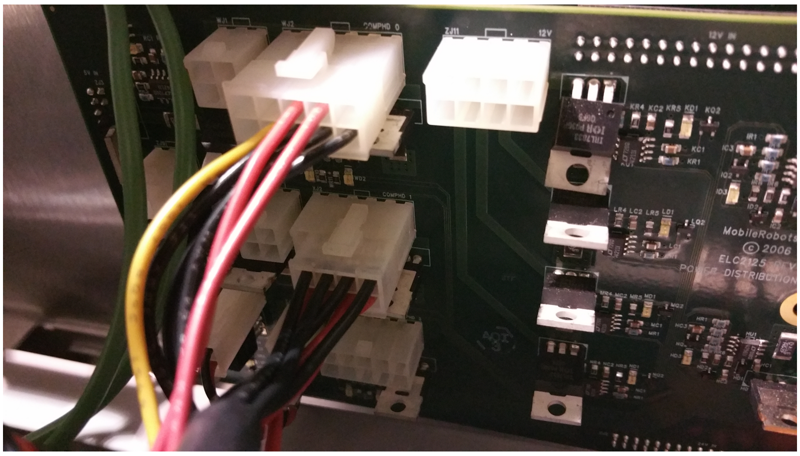


Figure 15: PDB Connectors on Seekur, top one is WJ2, lower one is XJ2

The +5V, +12V, and GND connections run from WJ2 to screw power terminals hammered into the wooden mounting panel on the top of the mounting bracket. When the robot is turned on, these terminals can be tested with a multimeter to ensure they are at their proper levels. Wires that are attached to these terminals have appropriate labels to indicate which terminal they are connected to (+5V, +12V, or GND).

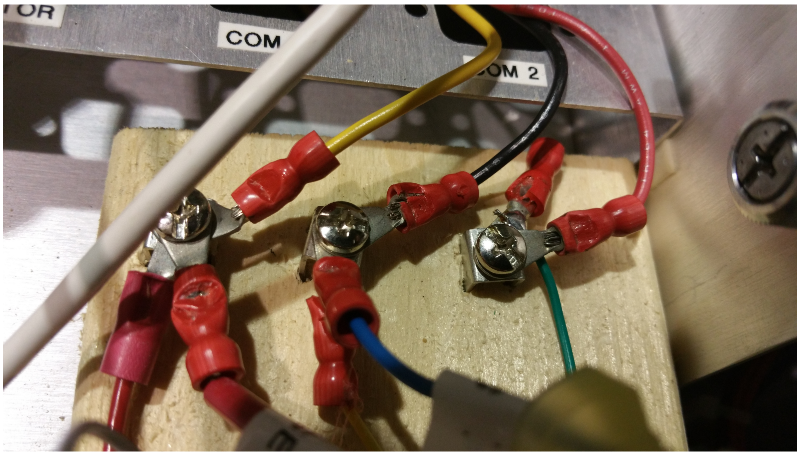


Figure 16: PDB power terminals

2.10.1 Relay Board

Ensure the intermediate relay is powered by connecting its labeled wire "+12V" to the 12V output from the PDB or screw terminals and connecting all GND connections to either PDB ground (for the small wires) or safety ground (for the large, pump (-) wire).

2.11 Maintenance

The top of the robot will need to be removed to perform maintenance operations. Once maintenance is complete, replace the top of the robot and attach all peripherals mounted on the top of the robot. This usually requires a team of at least three - two to hold up the robot top a short distance above the base while one securely fastens all wires.

2.11.1 Charging batteries

The batteries may be charged by connecting the battery charger's alligator clips to the red and black charging terminals mounted in the wooden panel shown below. Ensure the polarity is correct - red goes to red and black goes to black. These terminals are directly shorted to the series battery terminals (GND and +25.6V). The correct battery charger (currently in Mick West's lab) should be used to charge these batteries, although it is probably OK to use another charger, as long as appropriate voltage and current limits are observed. See the batteries' datasheet for more information.

2.11.2 Seekur Fuses

The fuse at the following location is prone to blow for unknown reasons. MobileRobots support confirmed that this fuse is prone to blow, and it had been a source of many issues with the robot initially - nothing powers if this fuse is blown. Replace with a 3A rated fuse, Digikey part link: http://www.digikey.com/product-detail/en/0451003.MRL/F2583CT-ND/813061.

Figure 17: Charging terminal panel

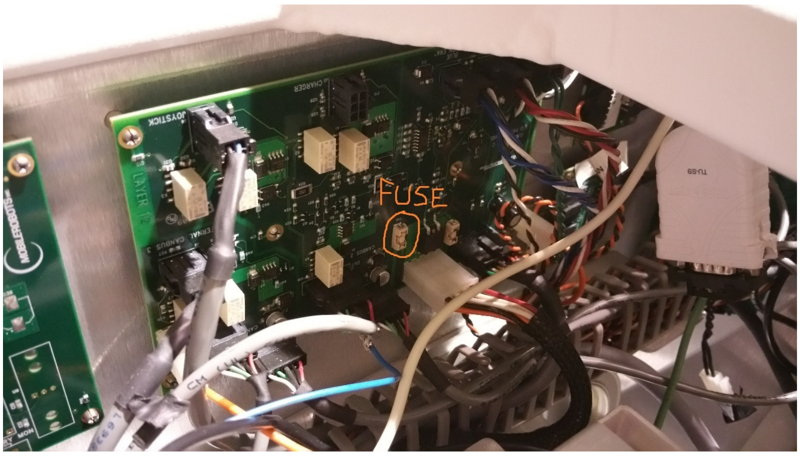


Figure 18: Location of capricious fuse

2.12 Powering On

Before turning the robot on, ensure the main safety switch is set to ON. Turn the robot on using the large metal ON button on the back of the robot. There should be a small clicking sound and the button should illuminate green.

Obviously, if there are any loud cracks or pops, something is likely wrong with wiring. Shut off the safety switch and debug the problem. Once the robot is on, it may be turned off immediately by pressing the OFF button. To drive the robot from software or the joystick, motors must first be enabled by pressing the MOTORS button and waiting 45 seconds. See the Seekur Jr reference manual for more details. If all wiring is correct, the Jetson should have booted up and connected to a local wireless network. Ensure the WiFi dongle is plugged into the Jetson, else it will not be able to connect. The Jetson is currently configured to automatically connect to the wireless network ”dlink” if it exists. Once the Jetson is booted and connected, one may run the ”ssh” program on a computer also connected to the dlink network. This will allow command-line access to the Jetson. The current username is ”ubuntu” with password

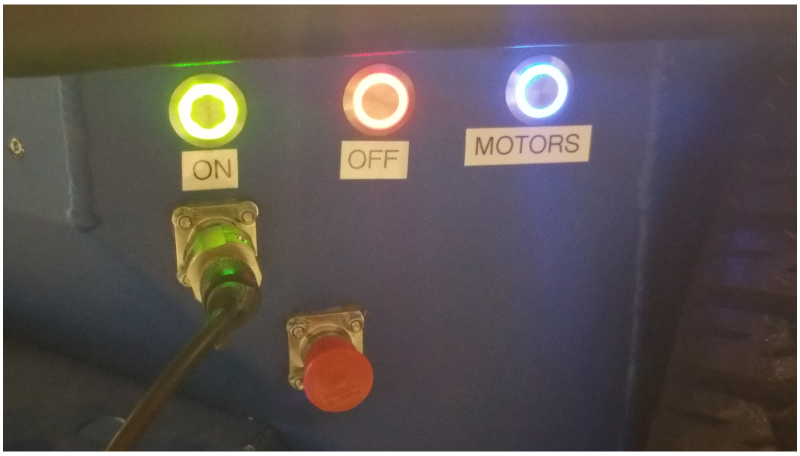


Figure 19: Robot powered on

"ubuntu".

3 Software Overview

3.1 Getting the code

The code, as well as CAD files for the project and more, are hosted in a private git repository. To receive access, any of the authors of this document should be contacted. A guide on

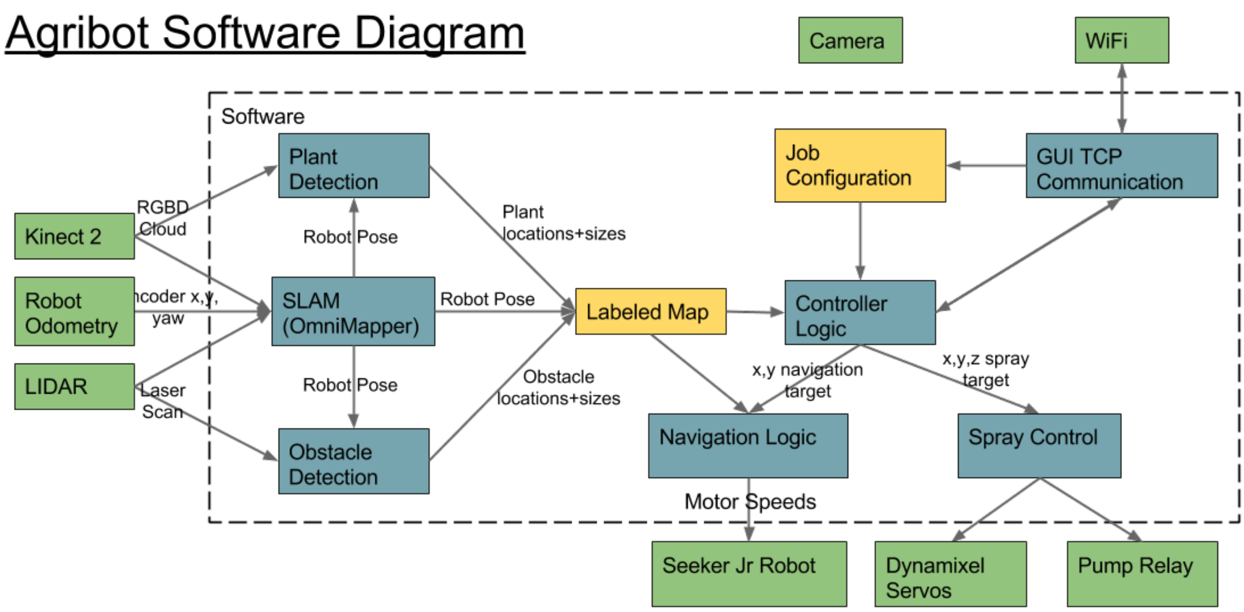


Figure 20: Software Diagram of Agribot.

using git is beyond the scope of this document, but can easily be found online. Many of the directories mentioned below assume you have accessed and cloned the repository to your hard drive. Since the repository uses submodules, ensure all submodules are cloned before attempting to build. A quick Google search will reveal how to clone all submodules within a repository.

3.1.1 Installing dependencies

- ROS

  ROS indigo is critical to the software of the robot. Follow the ROS installation instructions to set this up.

- ARIA

  The purpose of this library is to communicate with the Seekur Jr robot. To install, cd to code/JetsonTK1/lib/Aria and run ”make all” and then ”sudo make install”.

- MobileSim

  The purpose of this program is to provide a nice simulation environment for testing changes. It shows a graphical simulation of the robot moving around in 2D space, and ARIA has tight integration with it. ARIA will connect to MobileSim if MobileSim is running and no real robot is available on serial ports. To install, cd to code/JetsonTK1/lib/MobileSim-src-0.7.3. Ensure you have make, libtool, automake, and autoconf installed. Then run ”sudo make” and ”sudo make install”. MobileSim can subsequently be run by typing ”MobileSim” at the terminal prompt.

- OmniMapper

  See the OmniMapper git wiki for installation instructions.

- PCL

  Install PCL as stated above. Alternatively, following the PCL instructions for installing with sudo-apt get works.

- libfreenect2

  A specific fork of libfreenect2 is used for it to work with the Jetson. The instructions here work well, or the instructions on the git repo work as well.

- IAI_Kinect2

  This package is used to publish Kinect 2 data over ROS. After installing libreenect2 and confirming it works, just follow the official installation instructions.

- Sicktoolbox_wrapper

  This published the LIDAR data to ROS. Installing it as simple as sudo-apt get install ros-indigo-sicktoolbox-wrapper, once ROS is setup.

- base64

  The purpose of this library is to encode binary data within JSON strings. To install, cd to code/JetsonTK1/lib/libb64-1.2.1 and run make. A file named ”libb64.a” should be created in the src directory upon successful build.

- rapidjson

  The purpose of this library is parse and generate JSON for communication. It is a header-only library, and as such, no making is needed.

- serial

  Follow the README within the serial package, under code/JetsonTK1/lib/serial.

3.2 Compiling

The CMakeLists.txt file sets up most of the compilation settings, with package.xml containing specific ROS dependencies. To compile from scratch, use ”rosmake” on the directory containing CMakeLists. If this has been run, you may go to the build directory and type ”make” to recompile. To configure what is compiled, edit options at the top of the CMakeLists file.

3.3 Running

The file demomain.cpp, and all test files, can be run on command line like any executable. For the main code, -h can be used to see the command line settings.

3.4 navigation package

The navigation package contains controller.cpp, which has the high level controller that controlled moving and spraying. Currently, it is set to use basic turn and point maneuvers with an extensive state machine for moving through different phases of operation. It was planned to use OMPL (the Open Motion Planning Library) to implement more robust motion planning, but this was never completed.

3.5 slam package

This contains slam.cpp, as well as lidar.cpp. The former contains all ROS publishers and subscribers needed to perform SLAM with OmniMapper, as well as PCL code for detecting plants (once again, never tested). Once initialized, a SLAM-based labeled map is continually created, which the controller can be made to use. At present, since SLAM has not been fully tested the current dmeo has the controller use a hard-coded map.

The lidar.cpp file contains simple logic for using the LIDAR laser scan to detect if an obstacle is blocking the way within a meter of the robot. In the future, more extensive logic will be needed to precisely detect obstacles and place them within the map, but as of the writing of this document the simple obstacle detection has yet to be tested.

3.6 comm package

This package encapsulates the communication logic for interfacing with the GUI over WiFi and for communication with the connected serial devices (the GPS and mbed at present). The include file for serial_comm presents the straightforward interface for getting data from the mbed, sending it spray commands, and getting GPS data.

3.6.1 Live WiFi Communication

While the robot is running and navigating, it has the capability to send live updates back to the GUI. The robot can send its odometry data back. TCP communication must be enabled by not passing in the -noTCP switch when running the robot's main program. The communication over WiFi is done using the TCP protocol, and the following communication format is followed:

```
packet format from jetson -> pc
4byte packet signature 'pack'
4byte length field <len>
<json content of <len> bytes>
...repeat


json fields from jetson -> pc
<message>:
{
"gps" : <float>[2], //optional gps coords in degrees [lat, long]
"odometry" : [4 x 4 float], // optional SE(3) of current robot pose estimate
"image" : <encoded image (of some format), base64 string> // optional image from webcam
"lidar" : <lidar> // optional
}

<lidar> : {
"angleMin" : <float>, // radians
"angleMax" : <float>, // radians
"angleIncrement" : <float>, // radians
"distances" : <float>[] // meters
}


packet format from pc -> jetson
packet wrapper is the same

json fields from pc -> jetson
<message>: <config> | <command>
<config> : see config_format.jsonspec
<command>:
{
"command" : <pccommand>
}
<pccommand>:
"START" | "STOP"
```

3.6.2 Mbed Communication

Two operations are currently possible: telling the mbed to spray a target for a specified direction, and requesting data from the IMU, which is wired directly to the mbed. THe communication protocol is specified below. All baud rates are 9600bps.

```
****** jetson -> mbed communication
```

```
packets from jetson -> mbed consist of one header byte followed by a variable
number of data bytes
Header bytes are
0x01 -> IMU data request
Description: requests IMU data
Data bytes: none
0x02 -> Spray Command
Description: tells robot to spray a location relative to the sprayer frame.
    X is forward, Y is to the left, and Z is upwards towards the sky.
Data bytes:
Count: 16 bytes
Bytes offsets:
0 -> X (32-bit float, in meters)
4 -> Y
8 -> Z
12 -> duration (32-bit float, in seconds)
Acknowledgement: Upon receipt, the mbed sends a 1-byte ACK (0xAC) back to the jetson

****** mbed -> jetson communication
packets from mbed -> jetson consist of zero header bytes followed by a variable
number of data bytes
IMU data:
Data bytes:
Count: 24 bytes (6 32-bit floats)
Byte offsets:
0 -> Xaccel (in gs)
4 -> Yaccel
8 -> Zaccel
12 -> rollAccel (degrees/second)
16 -> pitchAccel
20 -> yawAccel
```

3.7 util package

The util package includes various utility files, the most important of which is geom.cpp. The geom file contains the structures and additional math implementation needed to perform navigation.

4 mbed

Refer to comm.h to see all of the individual functions, structs, definitions, and object declarations. Note that you will have to use the given compiler to make a .bin file to flash to the processor. This compiler is available at mbed.org.

4.1 Communication with Jetson

Although a simple Serial structure provides the correct functionality, the internal hardware FIFO buffers are only 16 Bytes in size. Since the servos take some time to execute, additional commands sent before completion might be pushed out of the buffer. To fix this, a software buffer is used through the MODSERIAL library to extend it to 256 Bytes. The program makes use of single byte reads in the form of getc(). This is due to the serial handler on the Linux OS, which was decided to implement uint8 unsigned char communication. To get a 32 bit float number, four bytes must be read consecutively into a single buffer and passed to the memcpy() function to convert it to a float. To send data, just reverse the process by using memcpy() to convert a float into a character array and send each byte in order over serial.

The mbed parses through all of the data it receives until either 0x01 or 0x02. The former will fetch the IMU data and send it over to the Jetson, while the latter begins the process of spraying. For the spray command, the mbed reads off four numbers off of the Rx buffer (x, y, z, and duration) and stores them in a spray_data struct. An ACK is sent back to the Jetson so that it can continue. The data is then passed to a function that uses inverse kinematics to transform this x/y/z into a theta/phi polar system based on offsets of the spray arm to the center of the robot. These two values are used to aim the servos in the final function as discussed in section 2.3.

4.2 servos

4.2.1 Setup

For Dynamixel servos, a command packet is sent out that writes or reads values from the internal memory. There are numerous things you can do with these servos, and there is a lot of feedback that you can get including current load and temperature. You can tweak torque and speed settings, and even switch between two modes: wheel and goal modes. To toy with these settings a command packet must be computed and sent out based on the guidelines listed on the datasheet. For the purposes of this project, there is a library available for the mbed that does three things: set the speed for wheel mode, set the goal for goal mode, and set the parameters. Only goal mode will be used, as accuracy is needed to spray the plants. To begin, hook up one servo to the DATA line. The baud rate is changeable, so make sure you set it correctly when declaring the AX12 object (by default, it should be 1000000). Use the member function included to change the ID. Disconnect and hook up the other servo to do the same, using a different number. The program assumes 1 and 2 as the ID, but any numbers can be used as long as the object declaration matches. After this is complete, the two servos can be put in serial without any worries.

4.2.2 Implementation

As good as everything sounds up to this point, there are a few catches. First, all write commands take long to execute because it waits for a response that is automatically generated by the servo to report any errors. Second, the servo fails to move about every 1 in 10 commands. Not much can be done about the first issue, but the latter is fixed by executing the same command 3-5 times in a row. This, of course, means more latency before more useful work can be done. An alternative may be to move and only resend if the position is incorrect (there is a function to check current position).
Now would be a good time to test the servos by themselves. The goal position works for integer degrees from 0 to 300 (center is at 150, and 0 is CCW looking at the front panel). To write both servos at the same time, use 0xFE as the ID for an AX12 object. Note that doing this negates waiting for a response packet.

4.3 Pump Control

Once the servos have moved to their specified location, the next step is to drive the pump to actually spray (water) to the target location. One GPIO pin, used to drive the pump control, is connected to a single node on the breadboard. This node uses two 1 micro Farad capacitors connected to ground and a 220 ohm resistor in series to increase the time constant of the circuit, thus eliminating the large power drain that occurs when turning off the pumps and forces the mbed to reset. A diode is placed between the end of the resistor and the control board to prevent feedback when spraying is complete. The pin is set as PWM allocated used as DigitalOut since a PWM pin can actually handle higher current. The pin is turned on and then the program waits for a duration specified by the received data (duration is the fourth parameter). Once complete, the pin is turned off and the function returns to main().

4.4 Mounting

For the current Seekur Jr. robot, a mounting bracket has been included in the front. Use a breadboard with an adhesive back, and simply place the mbed setup on the wooden board. The DMU can be placed in the middle of the robot near the battery (note that due to issues relating to real time operations and soft deadlines, the DMU was not actually mounted on the final robot system). For the RS232 converter, an optional mounting preference in order to snugly fit the cable is to solder it into the protoboard with the pump relays and run wires back to the mbed. The servos will be mounted on the arm as specified, and cables can be run to connect it to the robot's GND and +12V and the mbed data line (Tx/Rx serial).

4.5 Powering

++
The mbed will require external power, but luckily the robot provides a PDB with varying voltage levels. Connect the robot's common ground and a +5V source to the GND and VIN pins located on the upper-left pins of the mbed in order to power it whenever the robot turns on. The mbed automatically loads the last program put on in, so there is no additional programming necessary unless anything needs to be changed. In this case, turn off the robot and connect the mbed to a computer and download the .bin file. Press the button on the mbed to flash the newest program, and disconnect the computer.

4.6 IMU software

A simple enough library is provided to get the six data values from the DMU (bypassing all of the timing requirements of SPI). The result is a simple member function call, DMU.acceleration('x') for example to get the x acceleration. Pitch, roll, and yaw have their own functions (DMU.pitch()).

5 GUI

The GUI is written in java with JavaFX. The project is structured to have a view (GUI.java), a model (GuiModel.java), a host of other entities like popup windows and tools, and networking capabilities (SocketListener.java and PacketProcessor.java).

The GUI's GuiModel object, called model, is passed around to all windows and tools so these things can modify and update the model directly. Variables in the GuiModel are public to ease this process.

The code is located in the /code/AgribotGUI/ directory. It is structured as an IntelliJ project, but if you just want to compile and run the source from the command line, you can do this from within the src/ subdirectory. If you wish to understand what things do by reading the code, I warn you it is difficult to read, as it relies heavily on JavaFX. Some classes are javadoced; others are not.

I have designed the GUI with ToolTips and such in the hope it will be easy to understand. A few strange, subtle bugs persist. To begin, follow the instructions given in the back panel: enter GPS coordinates and a magnification level. If you are connected to the internet, the GUI should display a satellite image from Google. You can click and drag to move around this map and use the + and - buttons to zoom. To demark a field, select the rows tool from the left set of buttons. Click and drag to define an area. Edit the area in the window that appears. Click okay. To create a job, click the selection tool and then click somewhere within a field. Specify job rules in the window that appears, and click okay.

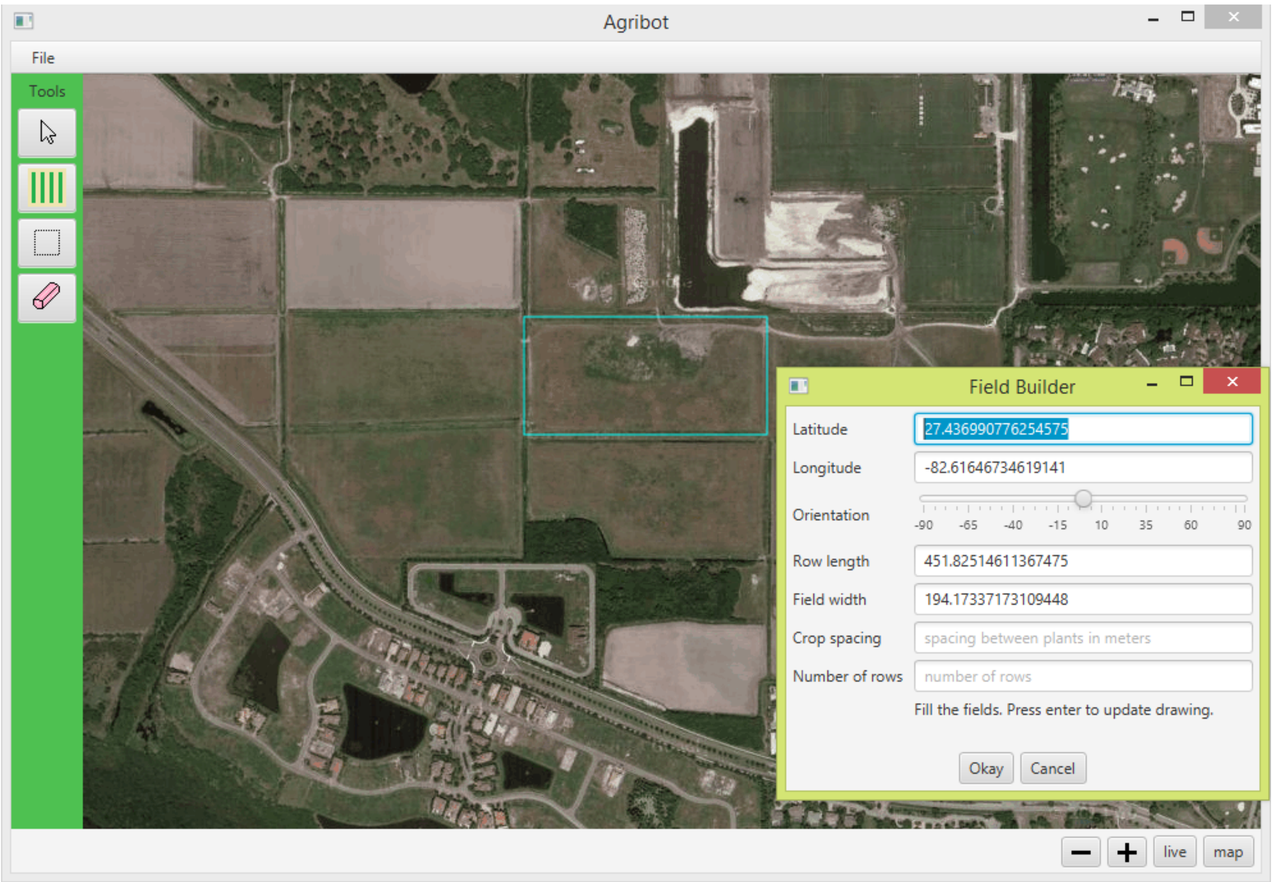


Figure 21: Screenshot of the map view of the Agribot GUI.

You now have all the information necessary to send some instructions to the robot. Select ”Set IP” from the menu and enter the robot’s IP address. If it is connected and running, you should see no error when you click okay. Now select the pointer tool again and click on the field you defined earlier. You should see some details about it in the window that appears. If you click the GenConfig button, the program will attempt to send the information you see to the robot. If this is successful, you should see no error message.

Now click on the live button at the bottom right of the GUI. This should take you to a screen where you can see some data as it is sent back from the robot every half-second. Note that not all data-displays are used because the robot software to send some of this data is not built. Note also that we originally wanted some live images displayed here, but this feature was not implemented because we could not get the Jetson to work with a camera.

I am sorry this is not so well written, it was kind of rushed so no nice lists or sections or formal technical writing here for a lot of it.

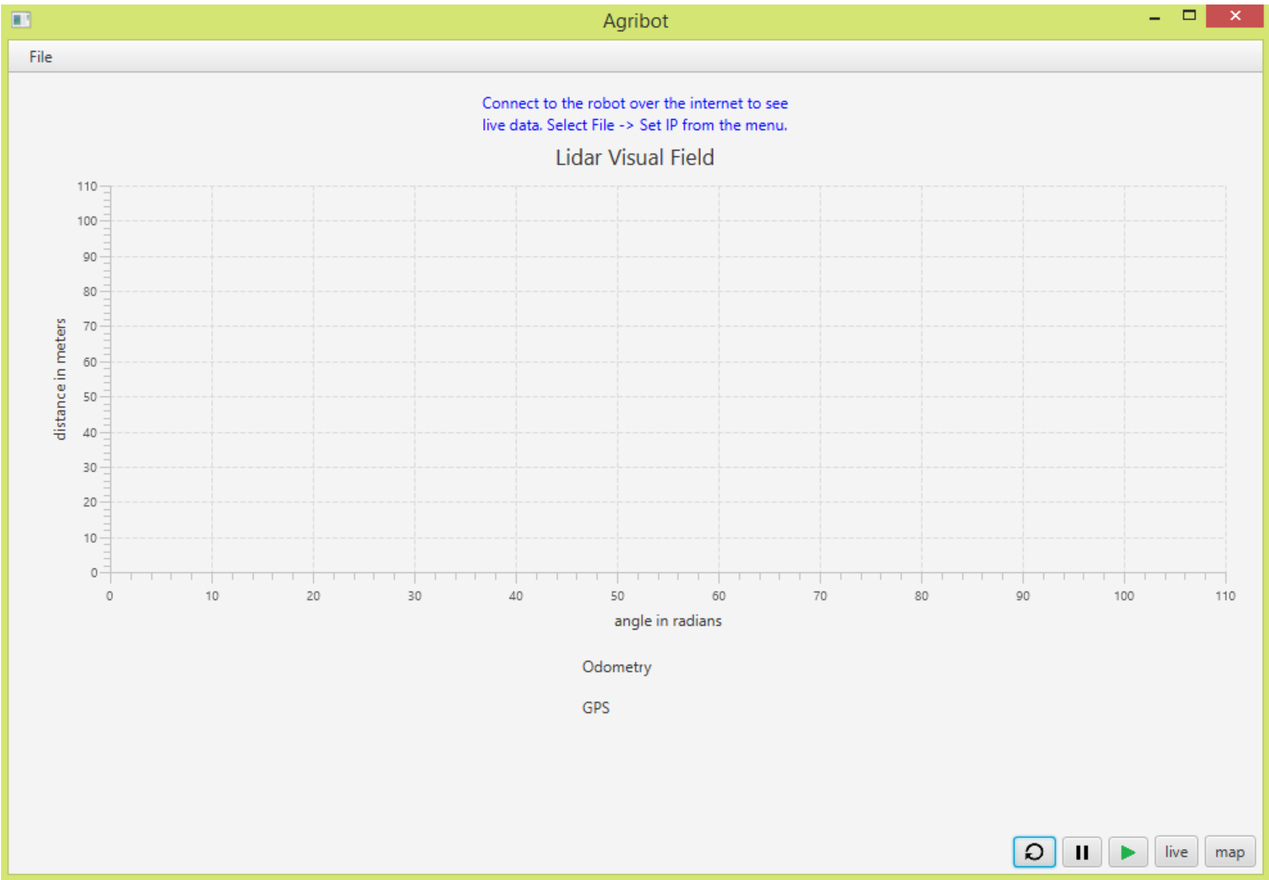


Figure 22: Screenshot of the live view of the Agribot GUI.

6 Demo Field

The demo field is composed of two groups of three wooden structures. These have been covered with false grape vines and grapes to provide an rough approximation of a row on a vineyard. The demo field was used for the final demonstration, and should be used for testing the SLAM and plant detection.