Project Number: P10216

ROBOTIC PLANT PLATFORM NAVIGATION

Alan Olson - EE, Mahbubul Alam - CE, Corey Provencher - EE, Marcus Gwillim - CE, Nicolas Bouret - EE

Abstract

The objective of this project is to build an autonomous, robotic platform that houses and cares for an on-board plant. In order to accomplish this, the project is divided into two parts, each with a specialized team. The goal of the Navigation team (P10216), is to have the robot gain an understanding of the plant’s well-being through various sensors. Since movement is required for this endeavor, the robot is also equipped with various sensors for collision detection and obstacle avoidance. The Navigation team interfaces the different sensors on multiple MSP430 microcontrollers, which are all controlled by a BeagleBoard Single Board Computer (SBC). The BeagleBoard also runs the main application software that navigates the robot and maintains a wifi connection to a host computer.

1. Introduction

In light of today’s movement towards a green environment, the concept of this project stems from the sustainability initiative at RIT. The idea of a robotic platform taking care of a plant is a novel idea, as the plant is a living thing. For millennia, human beings have taken care of other living things through sheer effort, and this is one of the few instances where something other than direct human interaction will directly shape a life.

This project is a brand new initiative, which will continue even after this Senior Design period. The P10215 team (herein by referred to as the Locomotion team), had the task of building the physical frame of the robot, as well as supplying it with energy. All of the artificial intelligence and control was assigned in turn to the Navigation team.

2. Needs

The robot has two basic requirements: it needs to be able to take care of the plant, and it needs to undergo safe movement. In order to take care of a plant, the robot has to have some understanding of the plants well-being. There are a variety of sensors that are used in tandem in order to provide the robot with this information. If the plant needs sunlight, then the robot will actively try and seek out sunlight. If the plant needs water, then the robot will spray some water into the soil. Thus, its primary functionality is to ensure the plant’s survival.

However, the robot must be able to move around, and thus the movement must ensure the safety of the robot, the plant, and any curious passerby. The robot is moving at a very slow speed of 17 ft/min, and thus the plant is not in danger. It uses a myriad of sensors in order to keep track of positioning, as well as detect incoming objects. Thus, the robots movement pattern has been optimized so that it does not hit any static (such as walls) or dynamic (such as people) objects.
3. System Level Specifications

The Navigation team is tasked to research different sensors in order to accomplish two tasks: take care of the plant, and provide safe movement for the robot. The following navigation sensors were chosen in order to provide the robot with a sense of its immediate geography: a GPS module, compass, accelerometer, four outdoor sonar sensors, and two indoor infrared sensors. The GPS module allows for the robot to keep track of its surroundings, and provides coordinates of the robot's whereabouts to the host computer. The compass is used to maintain a consistent sense of heading as wheel encoder counts rack up error. The accelerometer's main function is to act as part of the anti-theft system: if the robot were to be picked up, then the accelerometer would pick up a very quick acceleration, which triggers an alert to be sent to the host computer. The sonar sensors are used for collision detection to the front and rear of the robot, and the infrared sensors are used for picking up edges and drop-offs on the ground.

The other set of sensors that the Navigation team must interface with are plant sensors. This includes temperature, humidity, water level, and ambient light detection. The temperature sensor is used to predict when the plant may be in danger of freezing when it is too cold or drying out when it is too hot. Using a photosensor to detect ambient light conditions, sunshine or shade can be sought out according to temperature needs. A humidity sensor is used to determine if the soil is drying out and when to water the plant, while the water level sensor monitors the amount of water in the reservoir and sends a warning when it reaches critically low levels.

All of these sensors are driven by two Texas Instruments (TI) MSP430F1611 microcontrollers (hereafter referred to as MSP430s), and both microcontrollers act as slaves to the BeagleBoard SBC. The BeagleBoard runs the Angstrom Linux Operating System, and uses a USB WiFi adapter to connect to the RIT wireless network. It connects to both microcontrollers using I²C, communicates with the Motor Control Unit (MCU) using RS232, and communicates with the GPS module using a serial UART interface. All of the sensors are driven by firmware executing on the MSP430s, and their data is sent to specific register locations. The BeagleBoard then uses I²C in order to read in the register locations, and then store the values into its software. The main software running on the BeagleBoard is a Java application. The application is multi-threaded, and handles communication with the host computer, sensor readings, maintenance of other threads, serial communication between different devices, and intelligent decision making.

4. BeagleBoard

The choice of operating system for the BeagleBoard (pictured in Fig. 1) is made through testing the capabilities of the Angstrom, Ubuntu, and Android Operating Systems. Ubuntu is ruled out, as the version adapted for the BeagleBoard is found to only contain a command line interface with limited functionality. Android has a full GUI, but is designed for a touch screen interface and therefore does not have a suitable development environment. Angstrom proves to be the best choice, as the makers of the operating system have specifically adapted it to the BeagleBoard, as opposed to it being a third-party port.

Figure 1. BeagleBoard SBC

The next step in getting the BeagleBoard set up is...
to establish a connection to the RIT wireless network. While the wireless drivers do support WPA and WPA2 connections, the connection verification is limited to only passphrase codes and does not support the username and password format that the RIT network utilizes. Instead, a connection to the unsecured wireless RIT network is requested. While this is not the most ideal solution security-wise, it does allow for the BeagleBoard to connect to the wireless network.

5. Testing

A testing plan is drawn up to provide an organized way of proceeding through comprehensive testing procedures. This is important to ensure that each level of the design is valid before proceeding to the next level. Testing begins at the level of individual parts ordered from vendors and proceeds up through various levels of integration until the full system is tested.

Individual components are tested as they are received from vendors to ensure that no faulty parts are received and that their actual performance is up to what is demanded by this project. This is especially critical for sensors, which must be tested to confirm their transfer characteristic so that appropriate lookup tables or conversion functions may be generated for use in software. Proceeding upward from components, subsystems are tested to ensure that no unexpected interactions occur between components, and that the plans for firmware and software are valid. The full system test proceeds when all subsystems have been tested from both the navigation and locomotion teams, because a complete version of both the locomotion and navigation platforms must be available for a comprehensive software test.

Component testing for the most part proceeds nominally. Most components are functional within desired specifications, and all sensors have their transfer functions confirmed. A problem with the adjustable voltage regulator is encountered during power system testing, and more detail on this can be found in Section 6.5. Typical software testing and debugging efforts are carried out continuously during development, and problems are resolved on the fly. Firmware, sensor integration, and software testing are all carried out successfully.

6. Detailed Design

The detailed design of system components proceeds after a system level design is completed with considerations for the specifications. Design of the firmware which runs the microcontrollers is discussed in Section 6.1, the main software running high level threads on the SBC is discussed in Section 6.2, the navigation thread is explained in detail in Section 6.3, the setup of the sensors is explained in Section 6.4, and the power system for the navigation system is detailed in Section 6.5.

6.1. Firmware

In order to provide a sensor interface to the higher-level controls of the robot, the navigation sensors and plant sensors are interfaced to dedicated embedded systems that process incoming sensor data. The sensors are grouped in two main categories, plant and navigation sensors. Each group has a dedicated MSP430-powered embedded system for conditioning and recording sensor data.

Sensor data is written into registers locally on each MSP430. The registers are accessible to the BeagleBoard host-computer over an Inter-IC Communication (I2C) interface. I2C is a bus-oriented high-speed protocol that allows for significant bandwidth in delivering sensor data to the BeagleBoard. Each MSP430 connected to the I2C bus has a slave address. When the BeagleBoard needs to read a register from the MSP430, the address of the board is sent, followed by the register that is to be read. The MSP430 then clocks out the register value to the BeagleBoard. The BeagleBoard is the sole master of the bus and each sensor controller is a slave device.

The navigation sensors are polled at a 10 Hz frequency. This provides the BeagleBoard with data from each navigation sensor every 100 ms. The sensing cycle of the navigation sensors is shown in Fig. 3. The sequence of reading is designed such that the length of the cycle is minimized and the setup time of each sensor is respected. New sensor data is posted to the BeagleBoard by updating the new_sensor_data bit in the STATUS_REG I2C register.

The firmware required to realize this design is writ-
ten in C and compiled using TI’s Code Composer Essentials (CCE). The firmware takes care of all functions required to read each sensor and update the I²C data registers. The software flow chart in Fig. 2 shows the flow of one pass of the sensing cycle. At the end of the cycle, 100 ms has passed and the MSP430 re-enters the cycle to acquire additional data. The I²C data requests are interrupt-handled and thus do not disturb the overall sensing cycle. Figure 4 shows the overall flow chart for the entire navigation sensor firmware.

6.2. Software

The BeagleBoard is running on Angstrom Linux from a bootable SD card. A Java Virtual Machine (JVM) is installed in the operating system to provide support for the main application software. Wireless drivers are also installed in order to connect to the RIT network. Two main applications are created: The HostServer, and the RobotController. The HostServer is a multi-threaded server process that deals with connecting to the RobotController, and receiving/sending messages from/to it. The RobotController process is the main process that sits on the robot, and handles a variety of tasks. It is comprised of many threads that specialize in their various tasks. The NavigationSensor and PlantSensor threads continuously poll the MSP430 devices using I²C, and accordingly update variables stored in the NavigationData and PlantData classes respectively. The motor control data is stored in the MCUData object, and GPS information is stored in

Figure 3. Navigation Sensor Loop Timeline

Figure 4. Overall flow of navigation sensor firmware
the form of GPS objects. Other threads running simultaneously are the RobotServer and RobotClient threads, which specialize in connecting to and talking to the HostServer. Each spawns a temporary WorkerRunnable thread that takes care of a given Socket interaction. The AccelInterrupt thread checks for steep increases in acceleration (especially in the Z-axis, thus detecting that the robot is being picked up). The Maintenance thread checks sensor data, and either re-calibrates the necessary sensor, or sends an alert to the HostServer. The final thread is the main RobotController thread, which contains the main loop of the program.

The main loop of RobotController, shown in Fig. 5, which performs all necessary initializations, sets and stores the home position, then spawns all of the other threads. When all of these conditions are met, the RobotController thread enters a time based loop where it either wanders, takes care of the plant, or returns to its home position if it is out of bounds. Thus, all of these different threads are working in synergy to poll and maintain all of the robotic systems.

6.3. Navigation

The navigation portion of the robot control software is designed with dynamic object avoidance in mind. The primary objectives of the algorithm are to get the robot to a particular destination while avoiding any obstacles that are in the way. Since the robot will be active in mostly open areas, the majority of the obstacles will be people moving through and around the area in which the robot will be navigating.

The default state of the robot is to be moving forward, either towards a destination or to a randomly chosen location. When an object is detected at either the front sonar sensor, or at either of the two frontal ground IR sensors, the robot halts all movement and attempts to navigate around the obstacle.

Both the left and right sensors are checked for obstacles in order to determine which direction to turn. After turning, the robot moves forward until it detects no objects in the direction of its original path. After this, it turns back to its original angle and continues moving forward, aligning its turn angle to its destination if needed. If at any point during the obstacle avoidance phase another obstacle is detected in front of the robot, it halts its motion and restarts the obstacle avoidance process.

If both left and right sensors detect an object, the robot is forced to reverse until one of the sides is free, at which time it will continue as previously described. If an obstacle is detected at the back of the robot as well, the robot cannot make any valid movements and is forced to stop all motion. At this point, it waits for a pathway to clear and alerts the host computer if necessary. The complete navigation software flow chart is shown in Fig. 6.

The position and turn angle of the robot is determined using the onboard compass and GPS sensors as well as the encoder pulse counts supplied by the motor control unit. Using the encoder counts, the robot’s position is calculated and then combined with the readings of the GPS and compass with a weighted average.

6.4. Sensors

In order to be able to move and make decisions, the robot needs a set of specific sensors. In order to avoid obstacles, four sonar sensors are used. They are placed as follows: one on the front, one on the back and two facing to either side. This way, the robot is able to detect obstacles almost 360° around it. In order to detect edges on the ground and prevent the robot from falling, two down-looking infrared sensors are mounted on the front
Figure 6. Flow chart of navigation software loop
of the robot. Moreover, a GPS chip enables the robot to know its absolute location (this system is coupled with wheel encoders). In order to make decisions about the plant and itself, the robot needs to know some information about humidity, temperature, light, water level in the tank, and battery status. All of this data is available using appropriate sensors. The robot is also equipped with a 3-axis accelerometer and a digital compass that enable it to detect shocks and to know its orientation. A high level block diagram showing the component interconnections is shown in Fig. 7.

![Figure 7. Block diagram showing component connections](image)

All sensor information is collected by two MSP430s mounted on a perf-board. Sensors can be easily connected and disconnected thanks to Molex connectors. This perf-board is connected via I2C to the BeagleBoard and connected to a power board delivering 3.3 V and 5 V from the battery. The power board and the sensor board are mounted in an enclosure which is then mounted on the robot. This way, both boards are protected from external elements.

### 6.5. Power System

The power system for the robot is based on a 12 V lead-acid battery selected by the locomotion team to meet their needs for current drive and battery life. A system must be developed in cooperation with the locomotion team which adapts this main power supply to the power needs dictated by the sensors, controllers, and other modules selected for use to accomplish the design goals.

Among the parts selected, the majority require 5 V power, but some require no more than 3.3 V power. Therefore the 12 V main power must be converted down to at least two other lower voltage lines. Furthermore, the 3.3 V power bus is only driving a limited number of low-power devices, while the 5 V bus is driving a large number of more demanding devices. Because of this, the 5 V source must be able to drive significantly more current than the 3.3 V source. Both voltage sources must also be able to take in a noisy main power signal at a variety of levels and output a clean power signal at a specific regulated level.

The devices selected to meet these requirements are the Lineage Power 12V Pico TLynx 6A DC-DC Converter for the 5 V power and the Sure Electronics LM317 Adjustable Voltage Regulator for the 3.3 V power. After the power regulation hardware is received, initial testing is carried out to determine how precisely to integrate them into the system. It is determined that a resistance of approximately 1.3 kΩ must be attached to the Rtrim port of the DC-DC Regulator to set it to 5 V output. The DC-DC Regulator must also have a logical high voltage applied to its ON port in order for it to supply an output voltage. The output of the 3.3 V source can be used for this logic high value. With this setup, a constant 5 V output can be maintained with input voltages ranging from 6 V to 12 V.

Testing the Adjustable Voltage Regulator, a 6 V input is applied, and the on-board adjustable resistor is tuned to produce an output of 3.3 V. Attaching resistive loads to the output, it is discovered that a minimum current of 0.8 mA must be drawn in order to maintain a regulated 3.3 V signal. If less current than this is drawn, the output voltage increases to 4.4 V. Because of the low-power nature of the attached devices, this minimum current is not guaranteed at all times, and the devices could be damaged or shut down due to the resulting over-voltage. Therefore, a new voltage regulator is obtained and used instead: A TI μA78M00 Series 3.3 V Linear Regulator. It lacks adjustable output, but it has the benefit of being much smaller, easier to mount on a perf-board, and maintaining a constant output over a wider range of inputs.
With the parts selected and thoroughly tested, they are soldered onto a prototyping board according to the schematic shown in Fig. 8. Because both the 5 V supply and 3.3 V supply can maintain a constant output with inputs ranging from 6 V to 14 V, the power supply board’s $V_{in}$ may safely be anywhere in that range. This means it can either accept a power input from a 6 V regulator board provided by the locomotion team, or it can take power directly form the main 12 V battery.

7. Results and Conclusion

The components and subsystems are assembled and integrated together with the locomotion team’s system. This forms the complete robot hardware platform capable of movement and intelligent navigation. However, at this point it is not quite ready for final deployment. Other teams will take over to carry out final platform testing and further develop high-level software applications. Eventually, in addition to intelligent navigation and plant care, the robot will also be able to interact with people immediately around it and on social networks, thus making people interested in it and raising awareness for sustainability initiatives.

8. Acknowledgments

The authors of this paper wish to acknowledge Professor George Slack, Dr. Dan Phillips, Dr. Roy Czernekowski, Dr. Jim Vallino, Professor Stan Rickel, and Dr. Pratapa Reddy for the support they provided throughout this project, and the RIT College of Engineering, RIT Golisano Institute for Sustainability, and RIT Provosts Learning Innovations Grant for their financial backing.