SENSING IN COLLABORATIVE ASSEMBLY WITH UNCERTAINTY

Researcher: Devin McCulley
Virginia Tech, Field and Space Experimental Robotics (FASER) Laboratory
dmccu422@vt.edu

Advisor: Dr. Erik Komendera
komendera@vt.edu

Abstract

Automation in robotics needs a large amount of data which is gathered by sensors and data acquisition. This project intended to update the Mobile Assembly Robotic Collaborators (MARC) in the FASER lab to incorporate more sensors and a robust user interface. The use and general design of the prototype is described followed by the execution and results. The design was successful, and most components were tested but only a few were fully integrated into the MARC. Impact from the global pandemic led to slower than expected progress but more emphasis was put on the user interface instead. This leaves a lot of low-level code to be written and tested before there is an autonomous ready MARC.

Introduction

Problem Statement

This project aimed at creating a wheeled robot capable of semi-autonomous function and designed for future autonomous collaborative assembly research. The FASER lab's pair of custom mobile robot research platforms, named MARC, are constantly being upgraded to a new prototype version while also supporting several research projects. This report describes the process of making an incremental update to the MARCs that adds sensors, actuators, and a new user interface all needed for future autonomy research.

Starting Point

At the beginning of the project, the MARC was a Logitech F710 wireless gamepad driven SuperDroid platform with mecanum wheels and a six degree of freedom (DOF) GearWurx arm with no feedback except for battery voltage [1] [2]. Figure 1 shows the MARCs before the start of this project. The intended purpose of the MARC at that stage of development was to test the payload and reliability of a cheap off the shelf robotic arm and a robotic platform kit. The MARC was used to research mobile collaborative assembly through building office furniture and trusses as seen in Figure 2.

A chain of command is used to control the MARCs and have it achieve a desired pose. First the user inputs to a Logitech game controller which is picked up by an onboard

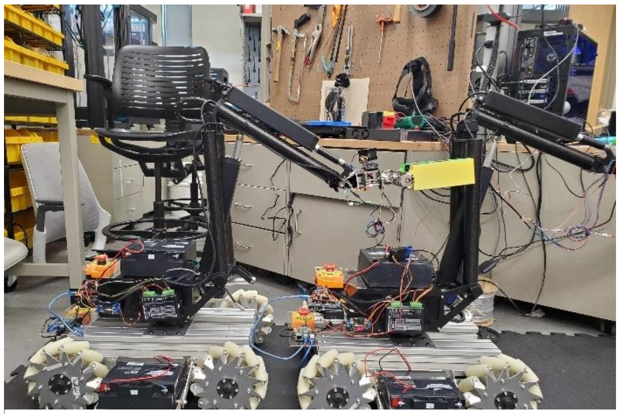


Figure 1: Initial state of both MARCs

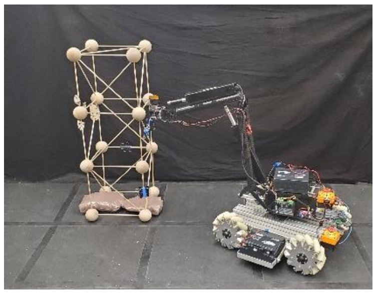


Figure 2: MARC deflecting a wood truss

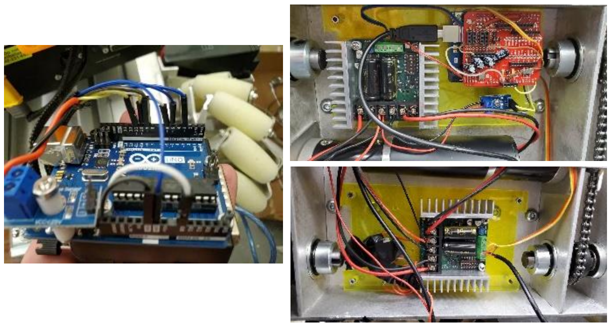


Figure 3: Arm control hardware (left) and platform control hardware (right)

Raspberry Pi 3b. The Raspberry Pi is running python and determines what state information to pass along to the subordinate devices based on the user's input mode. The Raspberry Pi passes along commands to the arm and platform's Arduino Uno that tells the desired state of the component. The initial state of the arm and platform control hardware can be seen in Figure 3. These commands are sent with UART through a USB connection and are formatted very simply for a quick response time. Components like grippers and wheels are controlled by velocity while the arm is controlled with joint angles.

The MARCs, in this early stage, had several limitations preventing progress towards autonomy. There was a need for feedback from both the arm and platform about the angle of joints and wheels as well as information about the immediate surroundings to avoid a crash. One downside of the mecanum wheels addressed in this project was the sliding that can happen because of the minimal contact with the ground and low friction. The robot base must be very stable when pushing or lifting a large amount of weight. The high-level Python code at the start of this project was only capable of driving the robot with a game controller. Any change to the USB port configuration for control hardware would require a minor code change and manually starting the python script once devices were connected and configured.

Changes to MARC

To be more capable of autonomy, this project added the following things: Teensy 4.0 microcontrollers, encoders to the platform wheels and the arm joints, six ultrasonic distance sensors, and four linear actuators to the platform [3]. The encoders provided angle measurements of all degrees of freedom. The ultrasonic sensors allowed detecting obstacles close to the platform and the linear actuators lifted the MARC off the ground and served as a brake. The positional feedback from the platform's linear actuators was also incorporated in this prototype upgrade.

The python code that previously allowed just controller input was modified to be a command line interface that would allow input through the controller, command line, or over wireless packets from another computer. This interface was also designed to show the user all relevant feedback information and adjust configuration settings without restarting the program. This new user interface was created to be very intuitive, helpful, and user friendly.

Uses for Updated MARC

The new MARC hardware and software changes are capable of operating with more input modes and provides more feedback. The intention of this prototype update was to allow autonomy when the feedback is provided to an external server that can command the robot. The FASER lab has an OptiTrack motion capture system that can be implemented for additional input to the autonomy server. [4]

Autonomy is needed to carry out more robotics research areas that align with NASA's mission directorates [5]. Completing mobile robot assembly without human input would be useful for building structures on other planets or in uncertain environments. The ability to have two MARCs opens collaborative autonomy research areas useful when objects are heavy or when a job requires two robots. The FASER lab intends to study task assignment with reinforcement learning so jobs and

Methods

The first main component of the project was upgrading the MARC with more sensors and a way to stop which is handled by either the embedded system on the arm or platform. The second addition with this project was the high-level change to the user interface code handled by the onboard computer.

Low-Level Changes

The hardware upgrade of more sensors and actuators required a redone circuit and modified embedded C++ code for both the arm and the platform. These changes also required the higher level python code to be modified to work with the new subordinate devices. To support more I/O, the embedded controller on the arm and platform was upgraded from an Arduino Uno to a PJRC Teensy 4.0.

The sensor chosen for the arm was an AS5600 position sensor encoder for the custom wrist because it provided a single analog signal to reduce the number of lines running along the length of the arm. The existing encoders on the arm joints were also utilized since the GearWurx arm has built in position control through standard servo input. The GearWurx Arm 3.0 has a DB-9 connector where the encoder pulse output could be accessed and provided to the microcontroller.

The sensors chosen for the platform were the HC-SR04 ultrasonic sensor and AS5048A high-resolution position sensor encoder. The ultrasonic sensors were mounted along the base with three on the front and three on the back. These provided a good general field of view for about one foot surrounding the platform and they are simple and cheap compared to LiDAR. The platform encoders were mounted next to each wheel axis to give position feedback after the DC motor turns a chain to drive each mecanum wheel.

The linear actuator for the platform was an Actuonix P16-P with feedback and an actuator stroke of 100mm. The feedback was needed so additional sensors wouldn't have to be incorporated in order to know the state of the actuator. A high gear ratio was selected because the four actuators must support the weight of the MARC and its payload. One necessary feature of these linear actuators is they keep their position when power is removed. This is required because if the MARC loses power, it will remain stable and not damage the actuators or MARC.

The Teensy 4.0 is a much more capable microcontroller than the Arduino Uno with 40 I/O pins including many PWM output, analog input, and communication ports. The processor is an ARM Cortex-M7 at 600 MHz which is fast enough to allow for a future MARC upgrade. The electronics were first tested on breadboards to get the code and connections working. Then a protoboard is used to mount all electronics and create wire connection points for each of the sensors or motors. The protoboard typically has to hold very simple components like a resistor or capacitor used in wiring the sensors but the linear actuators required a separate H-bridge chip. A dual H-bridge chip was used on each side of the platform to each control a pair of linear actuators.

For the arm, components are either located at the base or at the end of the arm.

The Teensy 4.0, motor controllers, battery, and connection to the arm's control port are all wrapped around the base of the arm. The custom gripper containing two stepper motors and a limit switch, encoders, and EPM are located about 3 feet from the base at the end of the arm. To centralize wires, a 25-conductor serial cable was run from the base to the end of the arm with DB-25 connectors used to keep all connections available in a standard format.

A 25-conductor serial cable was also used on the platform to connect the front and back electronics circuit. With the H-bridge and the large number of sensors being placed on the platform requiring a lot of connectors, it was decided to use two protoboards to distribute the electronics. A DB-25 connector attaches each end of the serial cable to the two protoboards. The protoboard towards the back of the MARC platform holds the Teensy 4.0 which is closer to the Raspberry Pi, making it easier to connect the two. The protoboards are suspended on acrylic laser cut sheets along with the motor controllers that fit into the bottom of the platform hollow sections.

The Arduino IDE is used to program the Teensy 4.0 microcontroller and object-oriented programming is used to structure the code. The flow chart in Figure 4 shows the initialization and then functions called during the main control loop for each device. The Teensy has a standard message format for input and output over UART that uses commas to separate values and a new line character to separate messages. The order of

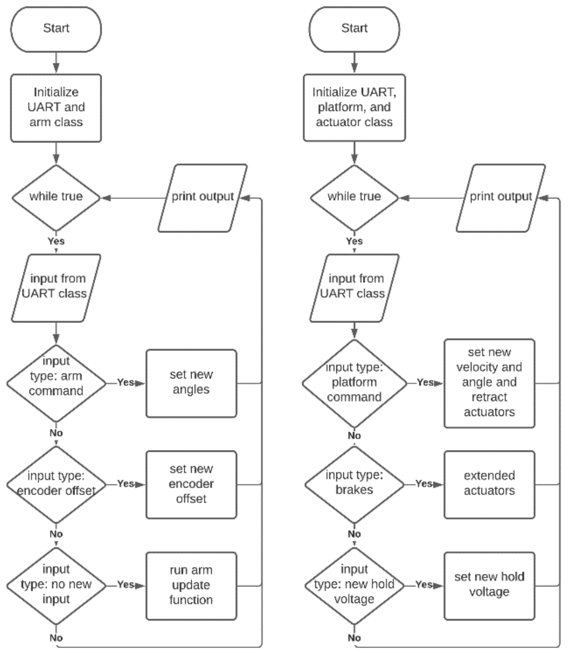


Figure 4: Broad view of embedded code structure for the arm(left) and platform(right)

values is predefined for both the arm and platform and shown in Figure 5.

Additional low-level code changes were done to make the platform and arm more configurable without having to reprogram them. On each device, there is a defined set of input types used to change the value of some variables. On the MARC arm, the encoder feedback can either return the raw values or the calculated angle at that joint. The angle offset used in that calculation can also be changed. The format for changing these variables is to send a command with a different prefix instead of "run." The additional available commands are,

	Input	Output
Arm	run,angle 1,angle 2,angle 3,angle 4,angle 5,angle 6\n	a,angle 1,angle 2,angle 3,angle 4,angle 5,battery voltage\n
Example	run,150,90,90,0,0,0\n	a,90,90,20,0,0,12\n
Platform	run,x velocity, y velocity, rotation angle\n	p,angle 1,angle 2,angle 3,angle 4,battery voltage, actuator 1,actuator 2,actuator 3,actuator 4,ultrasonic 1,ultrasonic 2,ultrasonic 3,ultrasonic 4\n
Example	run,120,0,0\n	p,10,30,10,30,24,200,200,200,200,4,4,5,10,3,5\n

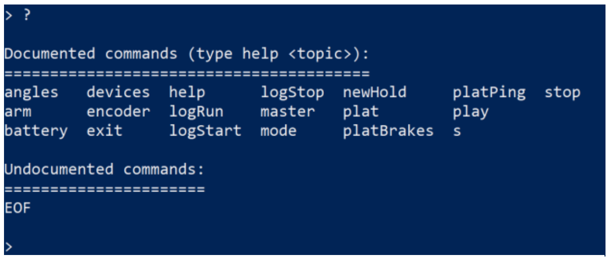
Figure 5: Arm and platform command format for state input and output with an example

"rawEncoder", "angleEncoder", and "encoderOffset." For changing the encoder offset, comma separated values of the new offset are required. Changing this value is intended for easy debugging purposes and could be used to easily apply device specific parameters stored on the controlling computer.

The platform has several new commands relating to the motors and linear actuators. The command "stop" will raise the platform and sending a usual "run" command will lower the platform. The command "newHold" can be used to set the motionless voltage input to the motor controller for each wheel. This voltage level can be different for each wheel and vary by device or current battery voltage, so it was useful to have a quick way to change the value. Each new command for both the arm and the platform must end in "\n" and provide any additional values with proper comma separating format.

<u>High-Level Changes</u>

The user interface was primarily designed to interactively handle multiple input sources to the MARC and display output information. The USB polling, command logging, and help feature were added to make the interface more useful and user friendly. The total list of available commands can be seen in Figure 6. The cmd library on python was imported and used as a template for building the user interface. [6] A thread is started in the code to read input from the command line and if the input matches with a defined function then that function is executed. A help function can also be defined using the same name as the input command.



```
> ?

Documented commands (type help <topic>):
========================================
angles   devices  help     logStop  newHold    platPing  stop
arm      encoder  logRun   master   plat       play
battery  exit     logStart mode     platBrakes s

Undocumented commands:
======================
EOF

>
```

Figure 6: Python command line interface available commands

This new interface supports modularity better by monitoring the USB ports to allow connecting and disconnecting devices without restarting the program. It keeps a running list of active devices and frequently checks if that device is still valid and looks for new devices. Devices are identified by either sending the letter 'a' or 'p' at the front of any output. The code uses this to assign a serial object associated with either the arm or the platform. The mode of operation allowed is dependent on what devices are connected. For example, if only the arm is connected then command line interface with the arm is allowable. If a gamepad and both devices are connected then the user will be given full control of the robot through that controller.

The controller mode of operation remained mostly the same as before this project began. A server mode of operation allows UDP packets containing commands for the arm and platform to be passed along to those devices. Interfacing the MARC through the command line is an option useful for debugging. In this mode, the user can specify a command to be sent to either device or choose from the list of configuration setting commands. The "arm" and "plat" command sends the provided line of text directly along to the appropriate device. The command "stop", for example, sends halt commands to both devices and prevents additional commands until "play" is entered.

The "devices" command lists if the arm, platform, controller, or server are connected. "Mode" will list the current operating mode and "master" will give operating preference to either the server or controller when both are available. The log related commands allow the user to save the device commands sent in a log file and re-send those exact commands keeping all timings the same. This is useful for testing the repeatability of certain sequences of moves that may be performed autonomously.

The feedback from devices is read by a thread and stored in arrays for when the user wants that information. Commands like "platPing", "platBrakes", "battery", and "angles" will display the most recent feedback related to that command.

Testing

A standard process of incremental testing was used in developing the hardware and software changes. The new feature is always implemented and tested in an isolated environment such as a breadboard or separate code file. Once the feature is completed, each possible use case is tested. When the hardware or software passes all tests and withstands most possible external errors, then it is integrated into the entire system. After integration, the entire MARC is tested ensuring the new and existing features work as intended. This process helps reduce bugs or flaws that can accumulate when new changes are put directly into implementation.

Results

Low-Level Changes

The circuit hardware was mostly completed for both the arm and the platform for a single MARC. Some additional changes to the MARC midway through the project was imposed by new FASER lab needs. This caused development time to be shifted from the original plan to implementing new features. These two new features located at the wrist of the MARC were a new degree of freedom and a new magnetic gripper. The additional DOF was added after the last joint on the GearWurx arm to allow for the rest of the gripper to rotate left and right. The second change was an alternative end effector that replaced the finger gripper with an EPM device that can create an electromagnet for attaching to a custom gripper on the device being picked up. The original gripper was kept as the main feature but the EPM also had all wiring required so the two could be interchanged.

Platform

The distribution of electronics between the front and back of the platform worked well with the 25-connector ribbon cable sitting flush along the bottom of the platform and secured to the metal supports. It is convenient to have two centralized points of connection for plugging in the actuators and ultrasonic sensors. The wiring would be messy if all components were connected to a single protoboard. The actuator H-bridge was a tight fit and the protoboard had to have some traces cut to make the connection point share rows with the H-bridge. There was a problem with a particular brand of wires used at first that kept breaking from the protoboard.

The encoders on the wheels were not fully integrated. Only two wheels had the encoders mounted and the hardware did not include the intended I2C multiplexer or the ability to connect that to the Teensy 4.0. The accompanying low-level code to read from the encoders and relay that as feedback was not written. The ultrasonic sensors and actuators were tested but not fully integrated. These features were not able to be fully integrated into the entire system because they caused delays that were unfavorable to necessary functions of the code. The physical mount for these features was designed but never completed so the actuators and ultrasonic sensors could not be mounted on the platform. The current state of the platform can be seen in Figure7.

The distributed H-bridges are convenient because where the actuators sit is close to the control hardware where they connect so the included wiring is long enough. For the ultrasonic sensors, the connections made near the protoboard are less than an inch long so cable extensions will have to be routed from the sensor to the appropriate protoboard. The DB-25 connectors on the platform were

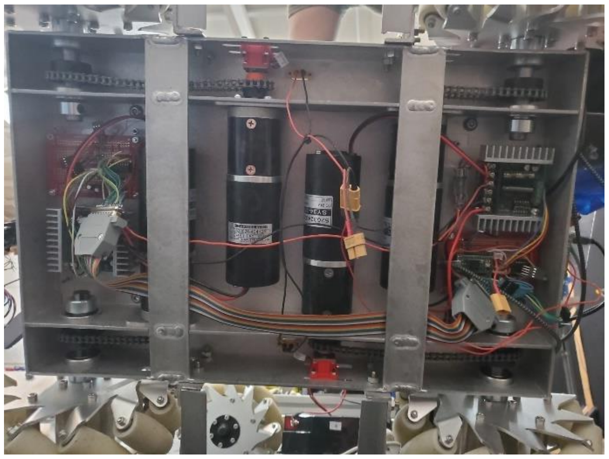


Figure 7: Platform with new Teensy 4.0 prototype circuit

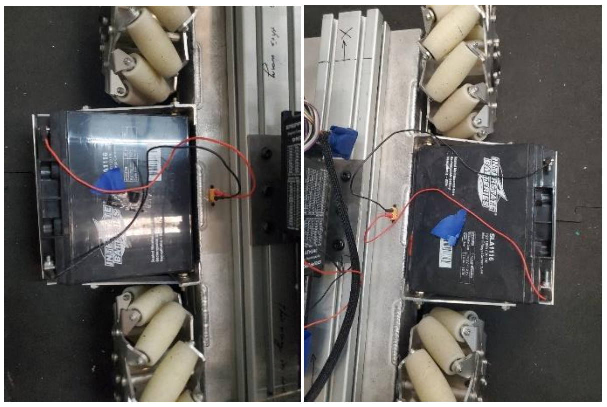


Figure 8: Surface mount battery connectors

bulky and could have been designed better to either be mounted on the protoboard or mounted somewhere more permanent. A better battery connection point was also an unplanned addition that can be seen in Figure 8.

Arm

A Teensy 4.0 update on the arm was successful and resulted in a clean protoboard full of connectors that either go to devices on the arm's base or head to the DB-25 connector to be routed up to the end of the arm. The current state of the arm control hardware can be seen in Figure 9. The DB-25 Connector is also used to send the output from the stepper motor controllers to the wrist and can be seen in figure 10. This design required some consideration about the current going through the long length of thin 28 AWG. To prevent too much voltage drop over the almost 3 feet length of wire, multiple conductors were used for some higher voltage lines. The 5V line used 2 and the 12V used 3. This use of additional wires to allow higher current and the large amount of existing control signals use all 25 conductors in the wire. To make connectors at the end of the wrist given that some voltage levels are carried in multiple wires, a small piece of a protoboard was used to solder common voltages together and create wires that go to connectors. The small board can be seen in Figure 11. Wires longer than a few inches going to or from the encoders and stepper motor drivers are put in cable shielding.

The encoders on the arm are available to the Teensy 4.0 through using a monitor splitter that uses the same DB-9 connector. This allows the encoders to be used by the arm itself and gives an easy access point without requiring splicing into wires. Code to read the encoders was tested in isolation but it was found too slow for use in the whole system. It is slowed by the pulseIn command in the Arduino library that waits for a signal to have a falling edge and then counts the time until the falling edge. This function can have the timeout value changed as a parameter but blocking the code to wait for encoder signals is too slow considering the rest of the program running. A thread running the pulseIn function or a custom non-blocking version of the function would be required to solve the problem. The encoder on the custom wrist joint was installed but later removed with the addition of the new DOF that can be seen in Figure 11. There was wiring and code changes made to support that encoder, but it was not tested or implemented.

An addition to the control hardware needed for the new joint was an H-bridge on the arm's protoboard in order to control the DC motor. The new DC motor has a hall effect sensor to provide position feedback. Some wires in the 25-conductor cable were

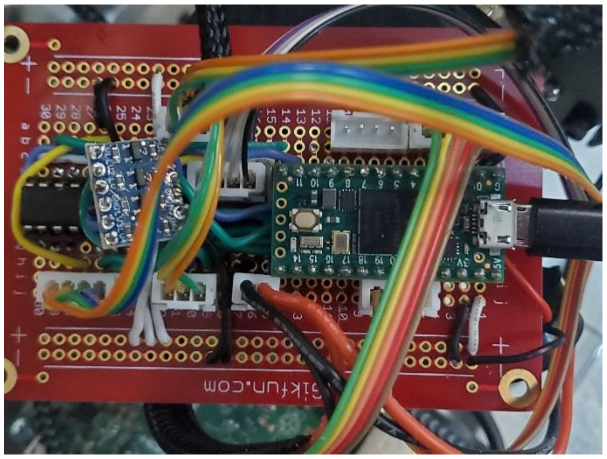


Figure 9: Arm's Teensy 4.0 control board with connections for all external devices

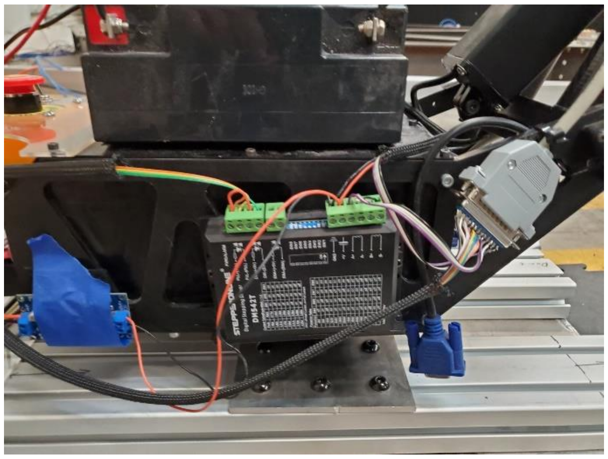


Figure 10: Side picture of the MARC with DB-25 at the base of the arm

dedicated to that DC motor control and feedback. This way of getting information to and from the wrist of the arm was successful and the new joint was fully integrated into this prototype upgrade of the MARC. The new degree of freedom required code change to the python so there is now a variable for the number of joints and the controller will behave according to what version of the MARC is being used. An additional feature was also added to easily switch between the old and new control scheme. This was done with a toggle of the Y button on the controller and was intended to help operators who are used to the original control schemes. The new joint was implemented fully and there is easy control of the gripper's rotation to the left and right.

An unplanned part of the switch from an Arduino Uno to Teensy 4.0 was that the stepper motor controller was not responsive because the Teensy 4.0 operates on a lower voltage of 3.3V. The stepper motor controller defines low voltage as anything below 4V which was not known originally so a logic level shifter had to be added in order for the two devices to successfully communicate. Since it was an unplanned addition, the logic shifter board is suspended above the H-bridge as seen in Figure 9.

Figure 12 shows the EPM gripper that required a UART and PWM connection ran up to the end of the arm. These were again done through the 25-conductor cable. The wiring and code were created and the device was tested but it was not fully integrated into the MARC. The connector on the MARC limit switch for the rail holding the gripper fingers was changed to a breadboard wire male and female connection instead of the previous connector that clipped in and was difficult to disconnect. The previous connector would have wires become disconnected often because of the force needed to separate the two.

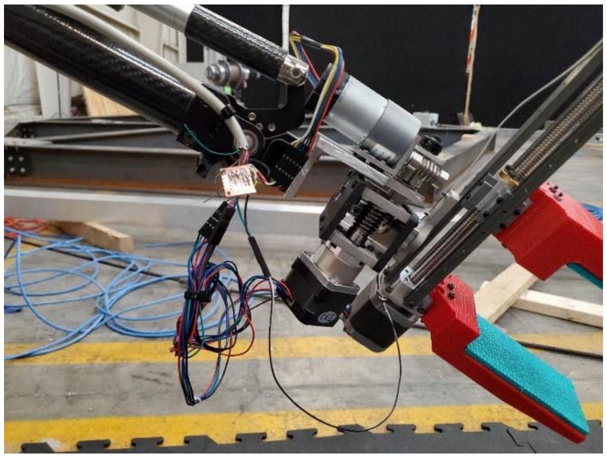


Figure 11: New DC motor joint and small board to handle wires

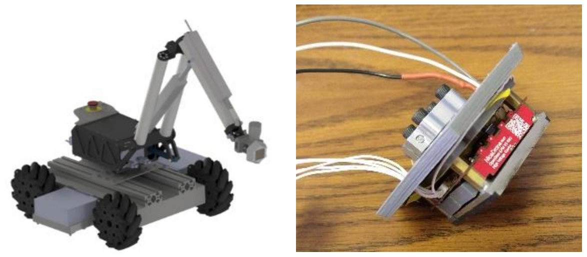


Figure 12: EPM gripper

High-Level Changes

The python user interface was a success and it allowed easy debugging and advanced use of the MARC but it was found less useful for regular use. The python script would startup automatically when the Raspberry Pi was powered on and the user would pick up the controller and start operating. Very rarely in our research did we need to connect to the computer wirelessly to open the command line interface. One new part of the upgrade used every time on startup is the USB polling. It is very convenient to have the Raspberry Pi recognize both devices and the controller and then automatically switch to controller mode. The feedback feature worked as planned but because the arm and platform weren't providing meaningful feedback, it was not useful. One down side of automatically starting the python script was that it would run in the background and the user interface could not be accessed unless it was terminated and started manually.

The logging feature was used minimally and the repeatability of most actions was low enough that it could not be trusted. The platform when given a simple trajectory to drive some small distance would end up in a different place after replaying the logged moves. The arm had similar troubles and because of the position control feature, the user would have to be careful to start and end the robot in the same pose when recording moves. This is because otherwise, the robot will snap from the end of the playback to where it started otherwise. While the server interaction with the MARC was not implemented, the updated prototype of python code was a successful step towards autonomy.

Discussion

Lessons Learned

There were several missteps in the design process and overlooked design details that prevented something from working. These delays include the need for a logic level shifter and the linear actuator mount in the back being blocked by the Raspberry Pi and emergency stop switch. Hardware limitations became a setback when the Coronavirus pandemic limited in person lab access. The user interface became a priority throughout the first semester of this project where there was no lab access. During the second semester, a lot of progress was made on one of the MARCs but that was slowed by other projects needing to use the MARC. There were many weeks where the MARC had to be fully operational so changes had to be very limited. Given the challenges and setbacks faced, the MARC still became more developed and moved away from the limited output on Arduino Unos. This project has overall improved the FASER lab's path to the goal of autonomous space assembly.

Future Work

The project has many components still needing to be integrated and tested. Once the MARC has finished getting all updates for this prototype, the autonomy server can be created and robots will be controlled from the server. The user interface will be more useful when the gamepad controller is used less and the server becomes the main form of controlling the MARC. At the same time as developing autonomy, the second MARC that is still on the first prototype can be reconstructed and upgraded to the second prototype as well.

With two MARCs the FASER lab can research autonomous collaborative assembly. The autonomous task assigning project can then be implemented on the MARCs to determine the best way about solving a given problem.

References

[1] SuperDroid Robots, "Mecanum Wheel Vectoring Robot - IG52 DB - DISCONTINUED," [Online]. Available: https://www.superdroidrobots.com/shop/item.aspx/mecanum-wheel-vectoring-robot-ig52-db/2063/. [Accessed 30 3 2021].

[2] GearWurx, "ARM 3.0 Long Reach Heavy Lift," [Online]. Available: https://gearwurx.com/product/robotic-arm-3-0-long-reach-heavy-lift/. [Accessed 30 3 2021].

[3] PJRC, "Teensy® 4.0 Development Board," [Online]. Available: https://www.pjrc.com/store/teensy40.html. [Accessed 30 3 2021].

[4] OptiTrack, "PrimeX 13," [Online]. Available: https://optitrack.com/cameras/primex-13/. [Accessed 30 3 2021].

[5] NASA, "NASA STEM Engagement: Overview," [Online]. Available: https://www.nasa.gov/stem/about.html. [Accessed 30 3 2021].

[6] Python, "cmd — Support for line-oriented command interpreters," [Online]. Available: https://docs.python.org/3/library/cmd.html. [Accessed 31 3 2021].