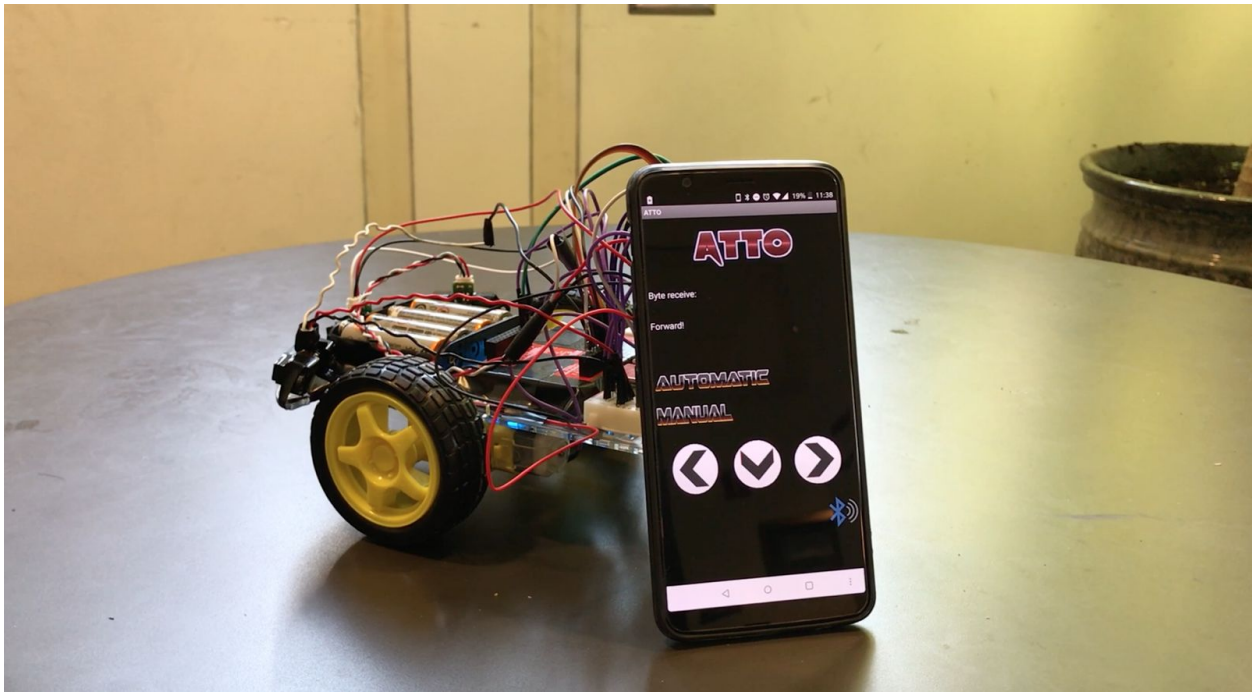
# EE 474
# Final Project Report

# ATTO




**Group 12:**
Andrew Comstock
Mert Akozcan
Zhuoming Zhang
Cloe Kyunghyun Lee

# Abstract

For the final project, our group decided to design a semi-autonomous vehicle system we call ATTO. ATTO is a remote controlled car that can detect and avoid obstacles. We designed this device to be the starter to larger projects like spacial mapping cars. ATTO was built up using smaller blocks of hardware and software, our major blocks were: range sensor, motors, motor controller, bluetooth, power distribution, user control application, and autonomous drive system. These blocks were combined to build and test ATTO.

# Introduction

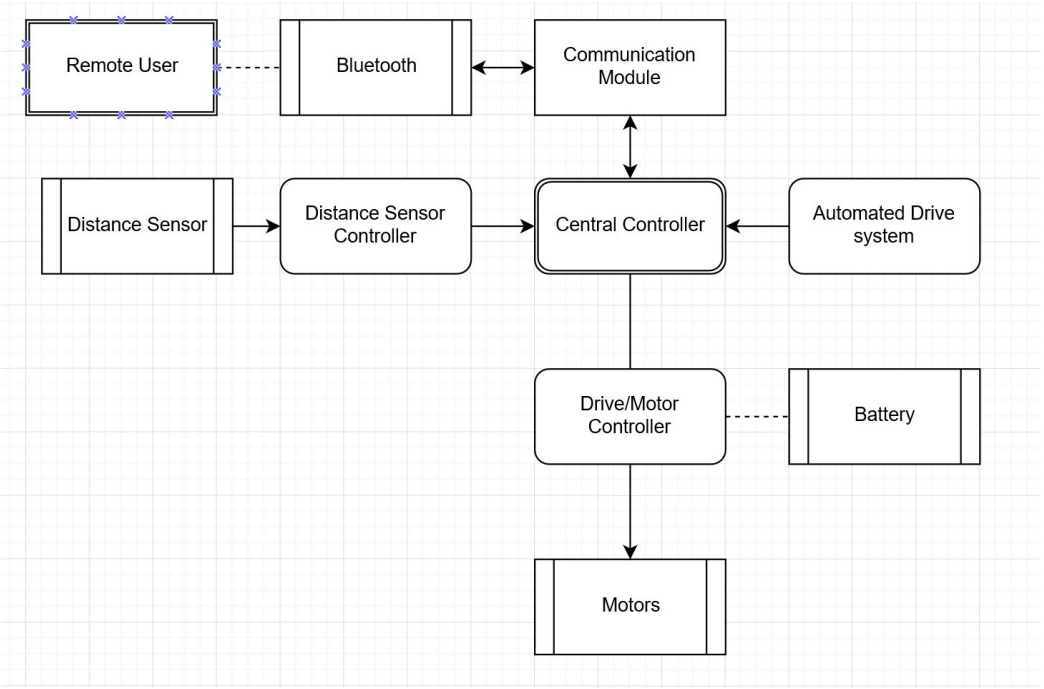
The scope of the ATTO project is to create the chassis, and basic hardware and software components of a self driving remote controlled car. The self driving features of ATTO can detect and avoid obstacles in its path and report status information to the end user via onboard bluetooth. The ATTO has independent bi-directional control over all its wheels, allowing it to move forward and backward, turn left and right, and vary the speed of its wheels. Using infrared sensors ATTO is able to detect obstacles directly in front, and on either flank. These sensors are connected to ATTO's main drive controller which is able to appropriately react to obstacles in its path.

The purpose of ATTO is to develop the introductory hardware for an autonomous spatial mapping drone, or similar device. The end goal of ATTO is to be put in a room with communication to a base computer and have it drive over the entire area of that room. This behavior can be used in applications like roombas, spatial mapping, and spatial problem solving robots.

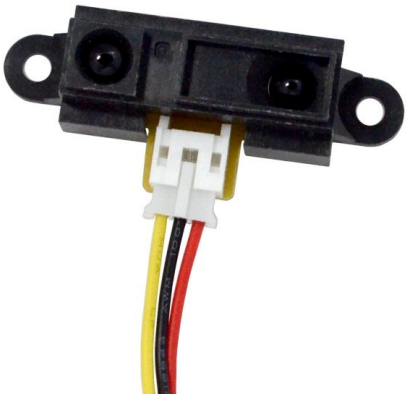
# Procedures

### Modules

You can see the component diagram below. We're using Tiva TM4C123GHPM as the central controller board. Our board is powered by a power bank. We have two motors, and to control them, we're using L298N motor driver. Our motor driver is connected to PWM and GPIO ports of the board. Our motors are powered from a battery case which includes 4 x 1.5V AA batteries. We have three distance sensors looking at forward, left and right directions. They're powered from board. We also have Bluetooth module to establish communication between the robot and the user by using Putty or Android app. Automated driving system gets the data from sensors and Bluetooth module, then processes available data to control driving. We'll now discuss all these components in detail.




## Range Sensor



We use an infrared distance sensor to determine distance of vehicles from obstacles. The model is Sharp GP2Y0A21YKoF Analog Distance Sensor, it has 10 to 80 cm range of view. The ADC converts a voltage signal within a given range to digital values to quantify the voltage's amplitudes. In order to calculate distance from ADC result, we used ADC Equation, "V = (x * 3.3v)/4095" and sensor equation, "distance = 27.86 * (1/(pow(volts, 1.15)))". We successfully measure the distance by configuring the ADC0 module. Three distance sensors are used to detect obstacles more precisely, and we initialized PE1(right), PE2(left), and PE3(front) for AIO input. After configuration, ADC Sample Sequence Control (SSC) 1,2, and 3 read values from input channel and take one sample at a time. When the board read flag from ADC Raw Interrupt Status, the results from Sample Sequencer are ready to use.

## Motor and Motor Controller

At first, we used two 12V DC brush motors and a L298 motor controller to control the two motors. We choose these motors because they provide 8600 rpm and high torque for ATTO to run. However, we realize the motors we

bought are not compatible with the chassis, so we have to use the motor comes with the chassis, which only require 6V and provide less power. In order to control the two motors' speed independently, we follow the data sheet and generate two PWM output from the Tiva board, and set the duty cycle for 75%. Then, we also use four GPIO pins to control the direction of the wheels independently so that ATTO can turn left and right by spinning the two wheels in opposite direction.

## Bluetooth Integration and Communication

By using Bluetooth communication, the device is able to communicate with an android app. This allows displaying information in an app, and a user can directly input instructions to the device. The Bluetooth model is  BlueSMiRF, it has 115,200 baud rate. We used UART for Bluetooth communication. We successfully send/receive data by configuring UART module 1, and this is connecting to pins, PB0(Rx) and PB1(Tx). In order to program a baud rate, we needed to use the baud rate equation given by the data sheet (page 903). we used integer value by, BRD = clk / (clkdiv * baud rate), and fraction value by, integer (BRDF * 64 + 0.5). When data is available to read/write, fifth(Rx) or sixth(Tx) bit of UART1_FR_R change 0 to 1. So then, we can simply use ReadChar()/PrintString(char*) function for Bluetooth communication.

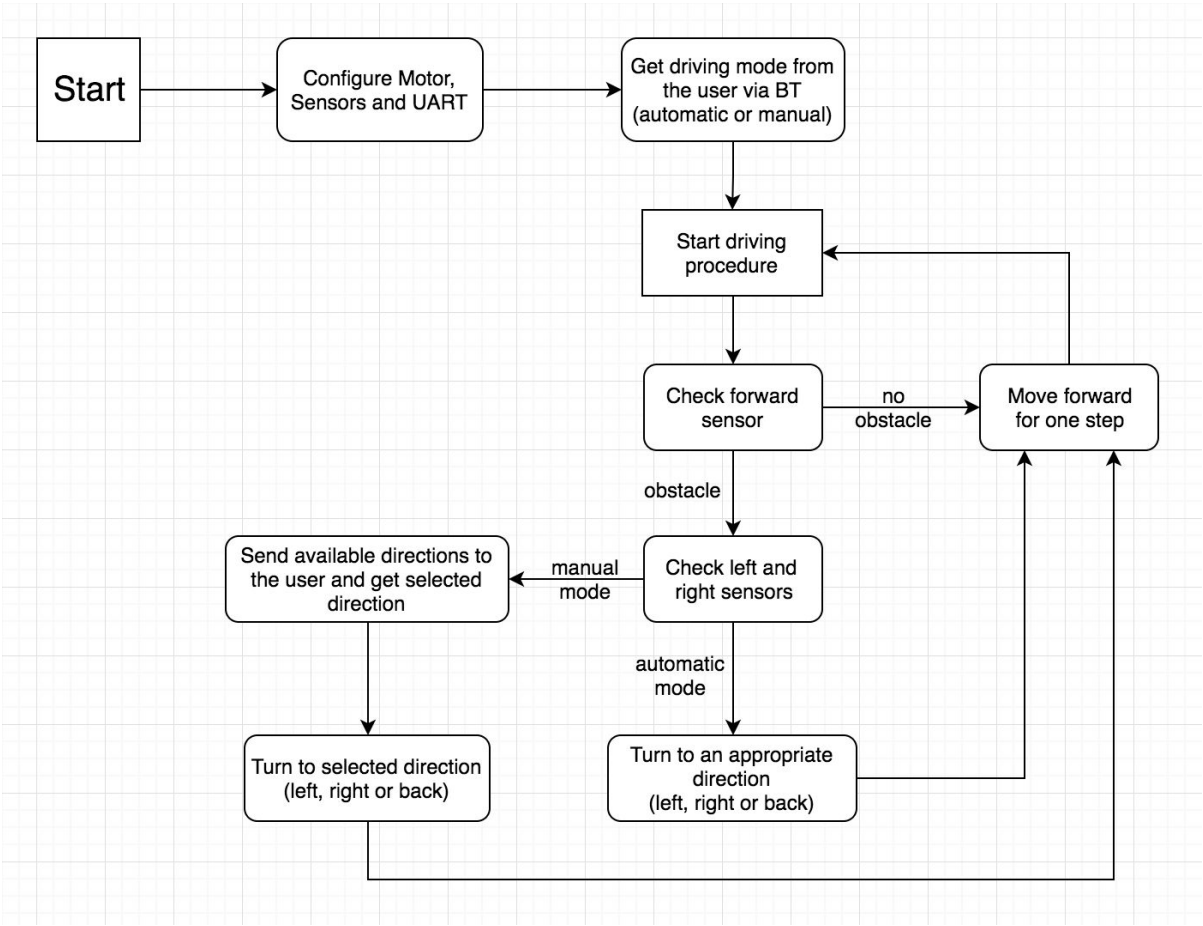
## Power Distribution

At the beginning, we use four AA batteries that have 6V to power the Motor controller. Since the L298N motor controller can provide 5V output voltage if the input voltage is higher than 5V, we connect the 5V output to the VBUS of the board. At the same time, we connect every ground together. Although the board only runs at 3.3V, the tiva board has an internal voltage regulator that can convert 5V input into 3.3V. In term of the three infrared distance sensors, we use the 3.3V from the tiva board to power them. However, this created a situation where the sensors and motors drain lot of power. The battery can only last few minutes. Then we observe a significant decreasing in speed. In order to solve this problem, we get another battery bank that has 5V output. We use the battery bank to power the board and the infrared sensor, and use the four AA batteries to power the motor. ATTO can last much longer than before.

## Chassis


We bought a premade chassis from Amazon, and we assembled it when we after testing all the individual component. There is a problem during assemble is that the motor we original bought is not compatible with the chassis. The 12V motor is slightly bigger than the 6V motor in the chassis. Therefore, we have to abandon the original plan. Instead, we use the 6V motor in the chassis. The advantage is that 6V motor will consume less power and the disadvantage is that the speed will not as high as the original plan.

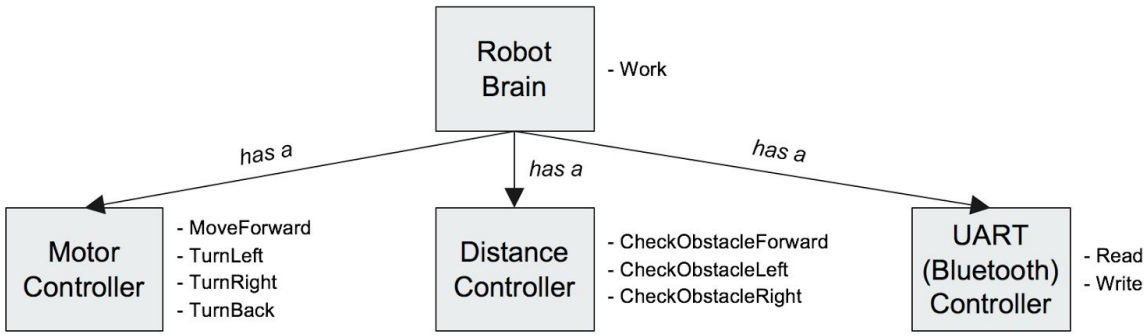
## Driving algorithm and main loop



You can see our algorithm above. Let's discuss it in detail. When our program starts its execution, it makes all necessary configurations by calling Configure methods of our controllers. We tried to give reasonable feedback to the user. After configuration, we send a message to user to select automatic or manual mode for the robot. We'll mention the details of these modes in a while. As robot gets running mode information from the user, it starts its actions. In each action loop, our robot controls the forward direction to check whether there is an obstacle. If there isn't any obstacle in forward direction, it moves forward until it sees an obstacle. When there is an obstacle in front of robot, it checks left and right sensors whether there is an obstacle. Now, running mode is important to proceed. If running mode is manual, robot sends available directions to user, and wants user to select a valid direction. After it gets direction from user, it turns in that direction and moves forward. If running mode is automatic, robot turns to a valid direction and moves forward in that direction. Notice that as long as there is no obstacle in forward direction, robot moves forward no matter what it's running mode is. This loop continues during lifetime of our program.

### Software overview

We implemented our project in C++ in order to benefit from its object oriented functionalities. There were few reasons we gave importance to OOP. The most important reason was to represent our physical objects such as motors, distance sensor, Bluetooth module in a natural way in our code as well. With object-oriented paradigm, we were able to build a virtual robot in our code. You can see high-level class diagram and public methods of those classes below.




*Classes and their public methods*

Let's look at the details of those classes.

We have a RobotBrain class that controls all the tasks and information for our robot. RobotBrain has access to every component of our robot, just like a human brain. It also controls the information flow between individual components of our robot. For example, the basic idea of having distance sensors is to send some information to motors. We send this concrete information through our RobotBrain, not directly from distance sensors (DistanceController) to motors(MotorController). With this structure we wanted our controller classes to be abstracted from each other and do their job. Our MotorController implementation has nothing to do with our

DistanceController implementation. Their connection established in our RobotBrain. We'll discuss driving algorithm and actual logic in RobotBrain in the next section.

- In MotorController, we configured PWM and Timer in register level. We implemented methods for actions that our robot can take. We can set the speed and direction of the motors inside this class. The reason behind using a timer is to specify the time interval our motors will work. In MoveForward method, our robot is set to move forward for half of its chassis length, and we ensured this using a timer. In turning methods (TurnLeft, TurnRight, TurnBack) we also specified certain time intervals with appropriate directions for our motors to work. Our robot turns approximately 90 degrees for left and right, 180 degrees for back.

- In DistanceController, we configured ADC in register level. We implemented methods to check obstacles in forward, left and right directions. Those methods turns boolean values, consistent with our design choices, it makes all necessary calculations on its own and sends the information we need.

- In UARTController, we configured UART registers. We implemented Write and Read methods as interface to our BrainController.

There is a *has-a* relation between our controllers and brain, consistent with strategy pattern in software design patterns. We also made our code public and open source by uploading it to Github.

### Mobile application

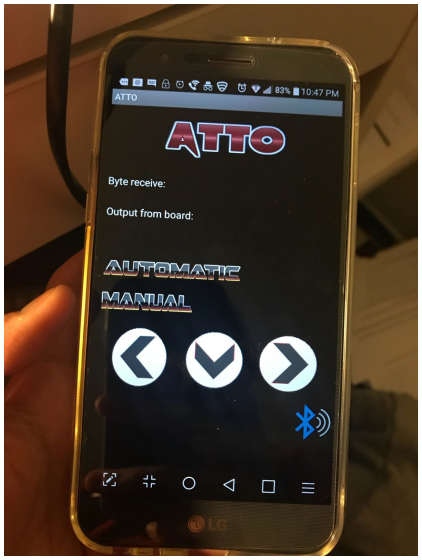
We didn't plan to make a mobile app for our robot at first. Our plan was to use Putty on a computer to establish communication between the robot and the user. After we were done with required part and Bluetooth module configuration, we had some time left, and we decided to go for a mobile app as an extra feature.

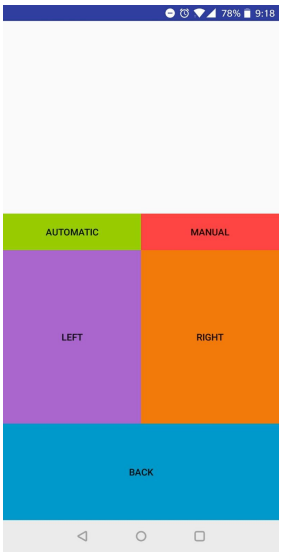
Mert tried to develop an iOS app by using Swift and XCode. Application was ready but because of schedule conflicts, we didn't have time to test iOS app with our robot during development process. After development process, we had a problem with connection between iPhone and our Bluetooth module as iPhone couldn't be able to connect to our module. Therefore, our iOS app became useless.

However, we didn't stop there. We were dedicated to develop a mobile app, so Cloe and Mert started working on an Android app separately, using different platforms. Cloe made an Android app by using MIT AppInventor, in 30 minutes, which had all desired functionalities such as sending and receiving data over Bluetooth. We were really impressed with AppInventor results. After testing functionalities, Cloe made improvements over user interface design. AppInventor offers drag-drop interface to develop apps. On AppInventor, one can develop application logic by putting certain blocks of elements like bluetooth connection and conditional statements

together, just like solving a puzzle. Also, designing UI is pretty simple with drag-drop interface of AppInventor.

Meanwhile, Mert was also working on an Android app by using Java and Android Studio. Our one intention was to showcase an app we coded together with our hardware code on Github. Mert was also partially successful with the app developed on Android Studio. One problem with that app was about missing data transfer. Some text (string) that comes from Bluetooth module to app sometimes had missing letters. There were well-organized tutorials about Bluetooth connection on official Android website. By following those tutorials, developing these basic functionalities wasn't hard. Although Android Studio helps with some basic requirements for projects, some important factors had to be taken care of by developer during development process. The most important one was multithreading. For an application like ours, we have to use multithreading, because we cannot block the main thread of our application when it tries to connect a Bluetooth device. This issue and few other minor issues were addressed in tutorials.

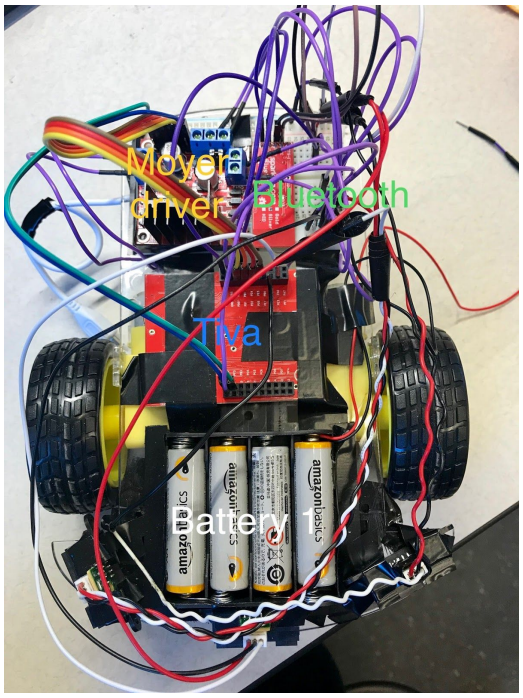


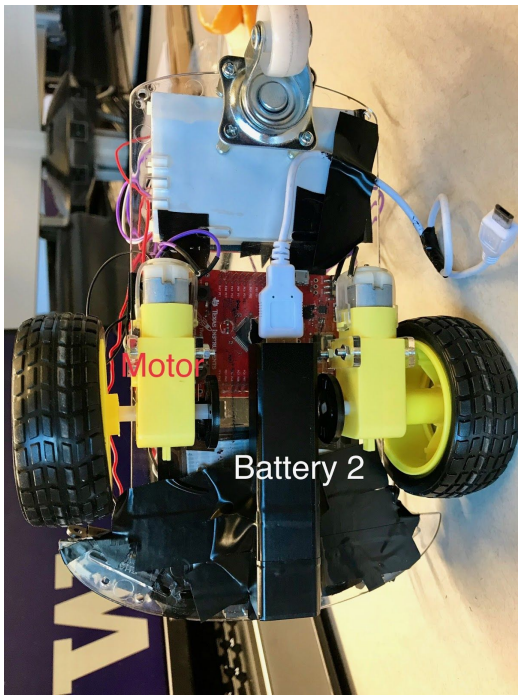




You can see the app developed with AppInventor on the left side, and Android Studio on the right side. They both have common functionalities. We can select driving mode as automatic or manual. We can display feedback information that comes from the robot in our apps. And of course, we can send direction commands if our robot is in manual mode. We were able to implement all functionalities that we had when we were communicating with our robot from Putty console.

## Assembly

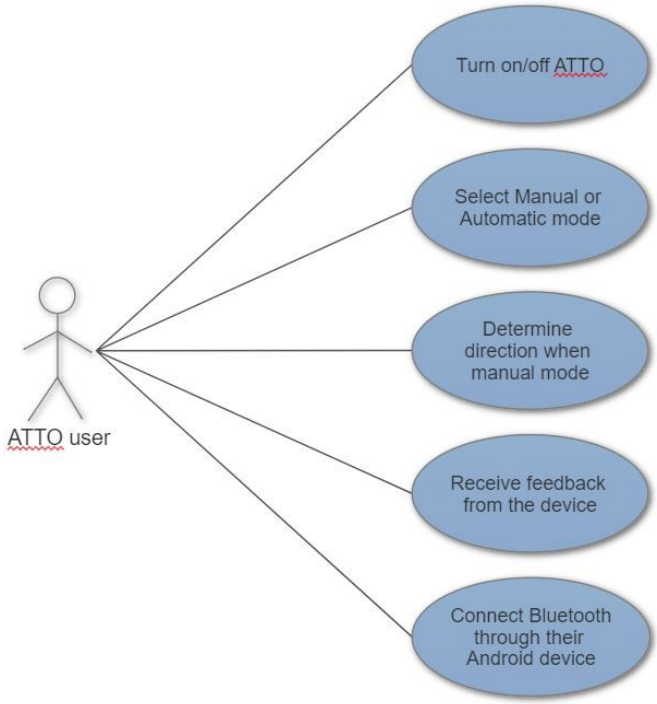
The following picture is how all the modules are connected together:

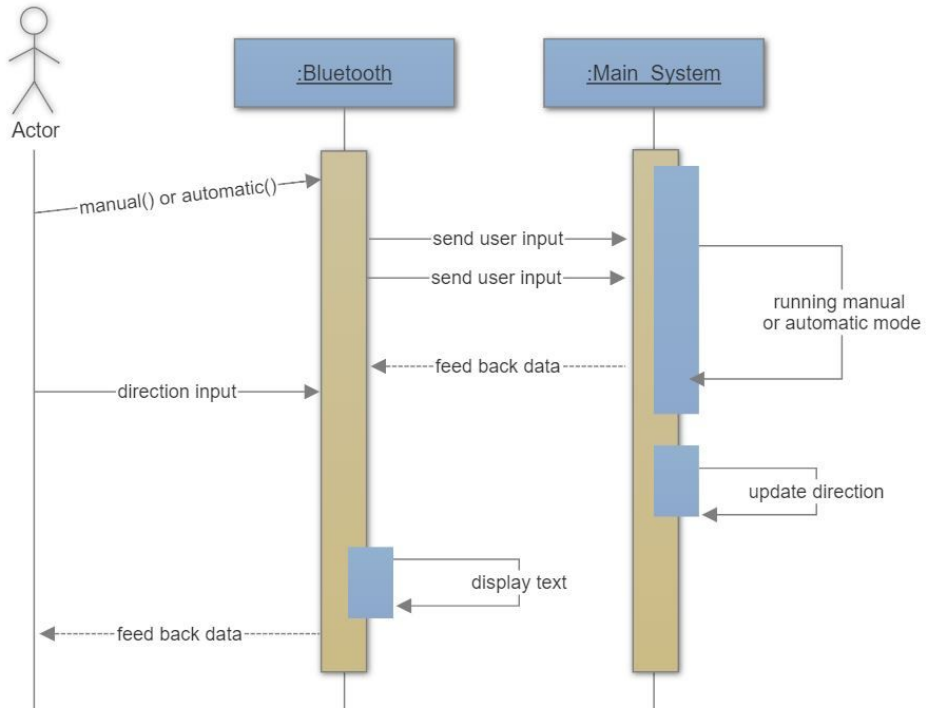



## Use Cases

The following diagrams are use cases and sequence charts. Use case describe possible applications of ATTO device, and sequence chart shows exchange of information between user and the device.






### RTOS

Our current system utilizes polling to switch between reading bluetooth input, writing bluetooth output, changing the speed and direction of the motor, and reading the distance sensor. This could be done a lot better by using RTOS to separate these tasks. Most of the CPU time of our program is spent idling waiting for a task to finish before it checks the next task. If we utilized RTOS, we could free up the CPU to compute other tasks while doing things like reading or printing to the bluetooth, or reading from the ADC. If we decide to continue ATTO to do more complicated tasks, switching to RTOS would be a requirement.

# Results

## Testing

Because we created everything individually and combined it later, our testing methodology was to test each individual component separate from all other systems to ensure correctness before integrating it to the overall design. All components were tested alone before being combined into the final ATTO device. As follows are the components that gave us trouble, and what we believe went wrong.

### Range Sensor

Through testing, we found that the range sensor would occasionally throw false positives, we are unsure as to why these false positives occur. They mostly occur when the sensor is pointed upwards towards the ceiling or other “complicated” objects like wires or pipes. The sensors also have a blind spot of under around 1-2 inches in front of the device.

### Motor Controller

In the testing phase we found that either the motor controller or the motor did not have the proper drive strength for ATTO. As such the device itself is rather slow, and rapidly drains battery power.

**Bluetooth**

We had a great deal of issue getting the bluetooth module to communicate between our TIVA board and PuTTY. There were multiple reasons for these issues. Our first issue was that the original computer we used to test bluetooth had driver errors that would not allow the bluetooth module to connect, which we interpreted initially as an error with the board, not the computer. Another issue we had was transmitting data through ports other than UART1 on the TIVA board. We eventually circumvented this by just using UART1, but never solved the initial issue. The final issue we had with the bluetooth was that we were unable to connect it to our first iteration of app for IOS. We found that this was due to apple restricting the methods of bluetooth communication allowed on the apple ecosystem.

# Conclusion

Through the creation of ATTO, our group gained a greater understanding of the individual components and how to integrate these into larger scale systems. ATTO has allowed each member of our group to expand their areas of knowledge outside of their comfort zone and learn new skills related to real world design and project management. Our final implementation of ATTO was able to drive around a room, modulate its speed and direction, communicate to a user with a mobile application, and detect and avoid obstacles. Through our testing, ATTO worked as expected.

# References


Tiva *TM4C123GH6PM Microcontroller Data Sheet*, *Texas Instruments*, Austin, TX, 2014.

Sharp GP2Y0A21YK0F IR *Infrared Distance Sensor* 10-80cm. [Online]. Available: https://www.bananarobotics.com/shop/Sharp-GP2Y0A21YK0F-IR-Distance-Sensor?gclid=EAIaIQobChMI9vKkkNKD2AIVlTRpCh3nXAVbEAQYASABEgI4jvD_BwE. [Accessed: 11-Dec-2017].

“SparkFun Bluetooth Modem - BlueSMiRF Silver,” WRL-12577 - *SparkFun Electronics.* [Online]. Available: https://www.sparkfun.com/products/12577. [Accessed: 11-Dec-2017].

“Bluetooth,” *Android Developers,* 02-Aug-2017. [Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth.html. [Accessed: 11-Dec-2017].