Abstract—With the growing number of robotics projects in academia and industry, the need for simple, generic, repurposable control is increasing. For example, the Machine Intelligence Laboratory in the computer engineering department at RIT is developing both an autonomous wheelchair and a people mover golf cart. As a precursor to autonomy, these platforms first develop remote control functionality to ensure all input and output devices are working properly. The current solution makes use of a third party Android application, Joystick Bluetooth Commander. This application works well for the team, but requires them to adjust their code to the non-standard protocol used by the application. Additionally, this means that they can’t easily switch applications. To solve this, we propose a new application designed for both Android and iOS, BlueBot Controller. This application provides a generic joystick experience operating on a simple two-way communication protocol over Bluetooth LE. The applications offer user customization to the interface such as the number and type of buttons on the screen along with a straightforward communication protocol. The development of these applications provides direct benefit to wireless applications, both for instructional usage in the classroom as well as research projects. Using a standard protocol in an easy to use package makes teaching, research and future support streamlined across several colleges and departments.

I. INTRODUCTION

The growing popularity of mobile robotics has brought with it an increase in demand for generic robot controllers. Our driving example is the Milpet team in the Machine Intelligence Laboratory of the computer engineering department at RIT. To control their robot Milpet, the team makes use of a third party application, Joystick Bluetooth Commander. This application runs on the Android mobile platform and provides simple controls such as a joystick and toggle buttons. These features work for the team, but at the cost of operation using proprietary commands over a Bluetooth connection, which poses a few problems. First, the application’s use of custom commands makes it challenging for the team to ensure their Bluetooth microcontroller is speaking the correct language required by the application. Second, the team is limited to the feature set provided by the application. Third, in the event that the team want to change applications, they need to learn a new protocol for a new application.

Other attempts at controlling mobile robots via Bluetooth have found that smartphones provide an excellent platform due to their availability and ease of use [1]. While it is clear that using smartphones is generally a good choice, the implementation of the control application can make or break the usability experience as seen by Gangan and Joglekar [2]. Due to the importance of usability, we aim to provide a simple interface to work with, both on a physical level and through communication protocols.

Our approach aims to start fresh with a simple IO protocol which supports two-way communication between a Bluetooth enabled smartphone and a microcontroller. The project develops two identical applications, one for Android and another for iOS. The applications are configurable allowing for a customized joystick, buttons, switches, and labels. The app communicates and transfers data via the latest Bluetooth LE standard, version 4.2. The goal is to make the application as generic as possible so it’s easy for anyone to work with.

The result of this approach is two applications which are freely available on their respective app stores that provide users with the generic joystick control they require. The applications, paired with a simple but powerful protocol, have been successful for the Milpet team in driving their mobile robot. Additionally, the application will be used as a teaching example for students in computer engineering classes such as Intro to Digital Electronics where the students learn how to connect a microcontroller and Bluetooth hardware module, and can communicate with it via the BlueBot Controller application. The following sections explain the creation and inner details of the BlueBot Controller application.

II. A SIMPLE DESIGN

One of the primary goals of the new application is to give the user the expected experience of a joystick. The joystick should feel intuitive, and also, transfer data in an expected and well defined manner. To achieve this, we utilize a simple two sided design for a landscape layout on a mobile device. By default, the right side contains the joystick itself, and the left side manages buttons and data fields. These two sides are separated down the middle by a joystick sensitivity slider which can be seen in Figure 1.

![Fig. 1. (left) The Android application. (right) The iOS application.](image)

As seen in Figure 1, both applications not only function exactly the same, but they appear visually similar too.
A. Joystick

One of the most important features of the application is the joystick itself. The joystick, seen in Figure 2, has two components. The first component is the black inner circle. This circle functions as a handle for the joystick. In order to interact with the joystick in any way, the user must first press within the joystick handle. The second component is the grey outer circle. This outer circle functions as a bounding region for the handle. The user is able to drag the handle anywhere within the area of the outer circle, but not beyond it.

Fig. 2. The circle bounded joystick.

The joystick will not respond if the user attempts to interact with it without first pressing on the inner handle circle. This prevents unwanted behavior where the joystick could move erratically from accidental presses on the far sides of the joystick background. Additionally, the joystick follows the user’s finger from their initial starting position. Rather than simply centering itself under the user’s finger, the joystick remembers the position that a press started from and remains centered under that point. While these features may not be critical, we feel that this implementation helps to more closely resemble a true physical joystick.

The shape of the bounding area is important because it dictates the underlying coordinate system used by the joystick. Since this joystick is bounded by a circle, a polar coordinate system is used. This is a change from the previous application which made use of a standard Cartesian coordinate system. Using a polar coordinate system follows logically since the joystick handle can only ever be a certain magnitude away from the center. Additionally, a Cartesian coordinate system can provide undefined results in corners.

With the polar coordinate system, $0^\circ$ is by default on the right side with $90^\circ$ facing upwards. Since this joystick is originally aimed at the Milpet wheelchair, and robots in general, the axis were rotated to have $0^\circ$ facing upwards. This would simulate expected behavior for driving, with an upward joystick command sending a message to go forward.

The other component of a polar coordinate system is the magnitude. The BlueBot Controller is set up to have a maximum magnitude of 100. This maximum magnitude is adjustable as described in section II-D. The output magnitude and angle of the joystick is displayed to the user which can be seen in Figure 5.

B. Buttons

Another method of input for the application comes in the form of buttons. By default the application launches with six buttons. This number is adjustable by the user, resulting in a choice of between zero and six. Each button has its own label and type which can be adjusted in the settings. Additionally, the application supports multi touch interaction so multiple buttons can be pressed at the same time in conjunction with joystick movements. An example layout of the buttons can be seen in Figure 3.

Fig. 3. A 6 button example layout.

There are two types of buttons available for use. The first is the default enabled momentary button. This button changes state when it is pressed on, and then returns back to the original off state when released. This style of button is more useful in a situation where it connects to a horn for example. The other type of button is a toggle button. This button changes state when the user presses and releases on it, and maintains that state until another press and release is received. This button is more useful in a situation where it controls lights such as Figure 3.

C. Data Fields

Receiving output from a Bluetooth module back to the phone is another key feature for the app. Like the buttons, the number of labels of the data fields are customizable from between zero and three. An example layout of the data fields can be seen in Figure 4.

The data fields are updated when a message is received from the connected Bluetooth module. The data fields are designed to display four byte floats.
The sensitivity slider allows the user to have on the fly access to the maximum magnitude of the joystick. The slider has ten values from 0.1-1.0. This slider can be useful in cases where the joystick is controlling a robot with powerful motors. In cases where the user wants to drive the robot slowly, they can reduce the maximum magnitude of the joystick. This keeps the magnitude changes based in the application instead of on the hardware side. The sensitivity slider can be seen in Figure 5. Figure 5 also shows the joystick coordinate details which displays the current joystick position in terms of magnitude and angle.

**E. Connection Status**

To give the user a quick understanding of the connection status, the area in the top left corner of the application displays a color coded connection message. An example of the app in the connected state can be seen in Figure 6.

Bluetooth LE is used for data communication in this application. Bluetooth LE provides power benefits over standard Bluetooth, and has support on both Android and iOS devices. A downside to using Bluetooth LE is that the underlying implementation is very different compared to standard Bluetooth.

Bluetooth LE follows a hierarchical design which can be seen in Figure 9. Each item in the hierarchy is defined by its 128 bit UUID [3]. This hierarchy puts a device at the top with a set of services below that. Each device can have multiple
services. Depending on the device it may implement some predefined services from the Bluetooth LE specification [4]. Alternatively, device manufacturers are free to add their own services. Each service contains a set of characteristics, where each characteristic contains a set of descriptors. Communication to a Bluetooth module’s serial port is not predefined by the Bluetooth LE standard, so device manufacturers have to choose their own service and characteristics.

In order to communicate with the serial port of a Bluetooth module, the app must know multiple details about the device hierarchy implementation. First the app must know which service to use. Secondly, the application must know which characteristic in that service is used for reading, and which characteristic is used for writing. Finally, the app needs to know which descriptor to use to enable device notifications. Since these details are not predefined and are not the same for every Bluetooth LE module, the application must be informed about the specific details beforehand.

Fig. 10. The BLEBee v2 module.

The Milpet wheelchair makes use of the BLEBee v2 Bluetooth module as seen in Figure 10 [5]. This specific Bluetooth module has three services defined. Only one of the services contains the characteristics used for writing and reading to the device’s serial port. In order to figure out which service should be used, we need to read up on the device specification. For example, the particular service of interest has a UUID of 067978AC-B59F-4EC9-9C09-2AB6E5BDAD0B [6].

Within that service there are four characteristics. The characteristics range from changing the serial baud rate to turning on the device LED. This device in particular has only one characteristic for both reading and writing, but other devices may have different characteristics for both. Within the read characteristic there is the notification descriptor. This is used to tell the module to notify the application when data is sent from the Bluetooth module to the app. Even more confusingly, some devices will only acknowledge that request if the app sends the notify byte array, while others respond to the indicate byte array.

What all these variations mean for the app is that, out of the box, it’s impossible to support all Bluetooth LE modules. Each module has its own UUIDs which the app can’t determine just from talking to the module. To deal with this, the application comes preloaded with the two Bluetooth modules that were used for this project, the BLEBee v2 and the Adafruit Bluefruit [7].

Fig. 11. The BLEBee v2 module.

Supporting only two Bluetooth modules would limit that value of the application significantly, so to deal with that we implemented the option for the user to add and edit additional Bluetooth modules. An example of this interface can be seen in Figure 11. Users can also switch between Bluetooth modules in the settings depending on the module that they want to work with. This feature allows the application to be used with many more Bluetooth devices.

IV. PLATFORM SPECIFICS

The applications were built from the ground up in both environments natively. The option to use a standard platform such as Unity or libgdx was available but was not chosen. As a result, the applications perform the same task and are visually similar, but are independently written. Each platform had its own unique problems and solutions which are described in the following sections.

A. Android

The Android application is written in Java and was developed with the Android Studio IDE. One of the primary challenges when developing the Android application was the Bluetooth LE support. Android has a fairly limited support stack for Bluetooth LE.

An example of this is the lack of message queuing. In order to write a byte array to a characteristic consider the last lines of Figure 12. We set the value of the characteristic to the bytes that we want to send. Next we write that characteristic to the device. This process is handled in a very simple manner by the Android back end. It’s a known issue that attempting to
write characteristics too quickly will result in the second write frequently being missed [8].

To solve this, we wait for the onCharacteristicWrite function to be called after a write has been successfully completed which releases the write lock. Additionally, to ensure that the write thread doesn’t loop unnecessarily, we use a blocking queue. This causes the thread to block until there is some data from the joystick or buttons to take from the queue.

Another challenge of working with Android is writing code for different Android versions. In Android 6+, API version 23, in order to turn on Bluetooth an application needs to have specific location permissions. Specifically, an application needs either the ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permissions. The BlueBot Controller supports Android API version 22 and up so we were required to check for device version in a few situations.

B. iOS

The iOS application is written in Swift 3 and was developed with the Xcode IDE. One of the primary challenges when developing the iOS application was the development of the settings menu. With no standard method for creating a nested settings menu, a custom approach was taken. By making use of a nested dictionary mimicking a table view we were able to create the nested settings menu desired. The nested dictionary allows for digging into the settings menu while pushing previous pages onto the view stack. Within the settings menu alert windows were used for simple text input such as button names. In more complex cases such a Bluetooth module details, new views were loaded in dynamically.

Another notable difference with the iOS development was the lack of use of the notification descriptor. On the Android side, this descriptor is used to enable notifications for messages coming from the Bluetooth module to the application. In iOS, this feature was not needed at all. See Figure 13 for the differences in implementation.

V. PROTOCOL

One of the main complaints with the application previously used by the Milpet team was, the data transfer protocol doesn’t make sense. The previous application used ASCII characters to send integer values. For example, sending a joystick value of 50, 100 would use six bytes in the form of: 0 5 0 10 0 0 [9]. This method is inefficient when two bytes could be used to display both numbers instead of six.

For sending data from the application to the Bluetooth module, the BlueBot Controller uses six bytes total. The byte array comes in the general form, | START_BYTE | OPCODE | MAGNITUDE | ANGLE1 | ANGLE2 | END_BYTE |. In cases where a button command is sent instead of a joystick command, the OPCODE changes, the MAGNITUDE and ANGLE bytes are replaced by the BUTTON_NUM and STATE, and a padding byte is used to keep outgoing messages the same length.

The START_BYTE and END_BYTE are 0x02 and 0x03 respectively. The OPCODE is either 0x01 for a joystick message or 0x02 for a button message. The magnitude is an integer between 0 and 100. The angle is an integer between 0 and 360 which is why it requires two bytes. The button number is an integer between 1 and 6. The button state is either 1 or 2.

For messages sent back to the application from the Bluetooth module seven bytes are used. The byte array comes in the form, | START_BYTE | OPCODE | DATA1 | DATA2 | DATA3 | DATA4 | END_BYTE |. The START_BYTE and END_BYTE are identical to the outgoing message and the OPCODE must be an integer of between 1 and 3 which represents the data field in which the data should be displayed. The four bytes allocated for data allows for float values to be sent back to the app. An example of sending data back to the application can be seen in Figure 14. This code is written for an Arduino Teensy in C++ [10].
This protocol is currently only used for the data fields in the application but the OPCODE field has set this up for future additions.

VI. USE IN THE CLASSROOM

While the primary client of the BlueBot Controller application is the Milpet team, the value that the project brings has planned usage in other areas of the campus. Specifically in CMPE-460, Interface and Digital Electronics. In this class, students will use the Adafruit Bluefruit module to enhance their understanding of serial communication [7]. Students will connect the app to the Bluetooth module to learn how to read and write serial data in an Arduino C++ environment.

To help with this process, another part of this project is a sample Arduino project that contains basic code to read and write data from the joystick app. This code reviews setting up a basic serial connection with a specified baud rate along with reading and parsing data.

VII. PUBLISHING THE APPLICATIONS

In order to make the applications as available as possible, we have aimed to publish them both to their respective application stores. The Android application is successfully published to the world and can be found on the Google Play Store [11].

iOS applications have a more complicated and expensive publishing process. The iOS application is currently available through the iOS Developer University Program [12]. This license does not allow the application to be published publicly to the iOS App Store, instead the application must be hosted by the creator. This is a temporary solution that will allow the application to be used in an academic setting by users with the application download link. Once funds are arranged for the payment of a standard iOS enterprise developer license, the application will be added to the regular App Store.

Both applications are available for free and aim to provide a simple joystick experience to promote learning.

VIII. CONCLUSION

The BlueBot Controller has accomplished the mission that it was created for. The Milpet team now operates with a standard and reliable Bluetooth protocol that can be run with either an Android or iOS device. The application’s various features such as adjustable data transmission rate and the customizable interface make it convenient to work with and easy to apply to different scenarios.

With both applications available for free and soon to be on both app stores, the BlueBot Controller provides value beyond the Milpet team to the community in general.

REFERENCES


