





# Robotics Society – Kart Workshop

## Session 3 – Programming

Last week, we wired up the Pico to the motor controller and connected the motors/servo. This week, we will be looking at the Pico programming, and how we can do some basic testing of the system.

**First as a note**, all Picos we provide will come pre-programmed with the controller code, and flashed with the MicroPython firmware. They will have a sticker taped to the bottom, this is the name they are advertising when you search for them in the PicoBot app.

MicroPico and Design files	PicoBot Bluetooth Android App
	

## Flashing firmware to the Pico

Hold down the BOOTSEL button while plugging the board into USB. The uf2 file below should then be copied to the USB mass storage device that appears. Once programming of the new firmware is complete the device will automatically reset and be ready for use. More information:

[https://micropython.org/download/RPI\\_PICO\\_W/](https://micropython.org/download/RPI_PICO_W/)



## Programming the Pico

We use [Visual Studio Code](#) to program the Picos, along with the [MicroPico extension](#). Be sure to check the requirements of the extension, especially the requirements for a Python 3.10 or newer version to be installed on your PATH or equivalent.



Clone the repo to a projects folder, or download it via the green box above.

Then, open the folder with VS code. If the MicroPico extension is already installed, it should automatically check your COM ports for a suitable device. When the device is plugged in, it will add a "Pico Connected" to your Status Bar, and give a notification that a connection has been made to the Pico. To upload your code to the Pico, press CTRL + SHIFT + P, or Mac/Linux equivalent, and search for MicroPico: Upload project to Pico. Select this option with ENTER, and wait for the code to be successfully uploaded to the device.

**Most importantly: Please keep the robot-X number the same as what is on the sticker on the bottom of your Pico! This will avoid multiple people ending up with the same advertising name.**



## Brief code explanation

While some basic comments are included in the code, an overview can be found below:

- The [aioble](#) library is used to handle bluetooth connections, specifically Bluetooth Low Energy (BLE) connections. When combined with the tasks below, it is able to advertise, and receive payloads from other devices (your mobile phone running the Android app).
- The [asyncio](#) library is used, which implements a subset of the standard CPython asyncio library. Three tasks are created to run concurrently in coroutines. These three tasks are:
  - **Blink Task** - This task simply checks whether a Bluetooth device is connected, and if it is, blinks the onboard Pico LED once every second. Otherwise, it blinks once every 1/4 second.
  - **Peripheral Task** - This task advertises the services and characteristics over BLE, and waits for a device to connect.
  - **Sensor Task** - This task waits for the chosen characteristic to be written to, and sets the motor control/servo control as required.
- There are then three additional functions. `control_motors` processes the angle and strength of the joystick input, and converts that to motor speed and servo angle values. `set_motor_speeds` directly modifies the PWM output of each of the pins, according to the values calculated in the `control_motors` function. `set_servo_angle` directly sets the PWM output for the servo control, as per the value calculated again in the `control_motors` function.

## PicoBot Android app

The PicoBot Android app is created as a joystick controller, and is built in Android Studio, and based off the PunchThrough android BLE guide. A short list of the files of interest to you are listed below. Other files are used, but contain the meat of BLE connections, and are not necessary to read through unless you want a deeper understanding of how the BLE connection is handled. Android apps are made up of many file types, but the two most important are Kotlin files and xml resource files. The Kotlin programming language is similar to Java, in that it runs within the JVM, but provides a more modern and concise language that is being adopted by more software developers. Using AndroidStudio, which is based on JetBrains IntelliJ, code linting and autocomplete is available, which helps with early Android development.

We make use of the virtual-joystick-android library, and a few other dependencies, to simplify the implementation.



File name	Description
<b>MainActivity</b>	Android apps are at their code made up of screens, called activities. Context, coroutines, and much more can be scoped to be available only to a specific activity. Diving in deeper, multiple fragments can be displayed within a single activity, but that is outwith the scope of this session. The MainActivity class in our app begins by requesting permissions if they have not already been requested, and sets up the scanning button, and recycler view (scrollable list that is resource efficient). When the Start Scan button is clicked, the app uses other classes to search for nearby advertising BLE devices.
<b>ConnectionManager</b>	Handles establishing, processing, and writing/reading communication from a BLE device.
<b>RobotControlActivity</b>	Creates and queues connections to be sent using the ConnectionManager to the Pico. Reads and parses the joystick input, and has TODO blocks where the special toggles can be implemented.
<b>activity_robot_control.xml</b>	An Android resource file, which creates the visual layout for the interface. It includes the joystick, logo, and special move buttons.