MAKE

Python Toolbox for Robotics Education

Carston Wiebe
cwiebe3@huskers.unl.edu

2024-07
Version 4.0

Make is a toolbox of functions and classes that mostly serves as a wrapper for CIRCUITPython and is used by the University of Nebraska Lincoln's PROTO RSO to aid in robotics education. The goal of the project is to create a psuedo-programming language that middle school/upper elementary aged students can use to code robots and learn simple programming skills.

REVISIONS

DATE	CHANGE(S)	AUTHOR(S)
2024-07-19	Proofreading	C. Wiebe
2024-04-29	Translated doc to LaTeX	C. Wiebe
2024-04-21	large_motor & small_motor	C. Wiebe
2024-04-12	button & functions	C. Wiebe
2024-04-10	PHILOSOPHY AND STANDARDS	C. Wiebe
2024-04-06	Techdoc creation	C. Wiebe

Contents

1 INTRODUCTION **4**
1.1 PURPOSE . . . 4
1.2 DESCRIPTION . . . 4
1.3 PHILOSOPHY AND STANDARDS . . . 4

2 BAPCAT **6**

3 CONTENT **7**
3.1 CLASSES . . . 7
3.1.1 button . . . 7
3.1.2 distance_sensor . . . 7
3.1.3 large_motor . . . 7
3.1.4 small_motor . . . 8
3.1.5 wagon . . . 8
3.2 FUNCTIONS . . . 8
3.2.1 pause . . . 8
3.2.2 until . . . 8
3.3 CONSTANTS . . . 9
3.3.1 BUTTON_PIN . . . 9
3.3.2 DC_PIN . . . 9
3.3.3 FRQ . . . 9
3.3.4 GROVE_PIN . . . 9
3.3.5 reversed . . . 9
3.3.6 SERVO_PIN . . . 9

1 INTRODUCTION

1.1 PURPOSE

Created for the University of Nebraska-Lincoln's PROTO student organization with the goal of providing a psuedo programming language that can be used by middle and elementary school aged students.

1.2 DESCRIPTION

Wrapper for CIRCUITPython (also written in Python) to be used on the MakerPI RP2040 and PROTO's componants. Branches out into creating unique functions and classes, but mostly serves to simplify existing code in both the standard Python toolset and in CIRCUITPython.

1.3 PHILOSOPHY AND STANDARDS

A minimum amount of knowledge on coding should be assumed and expected, and the users may not have access to a proper IDE with features like intelligent syntax highlighting or auto-complete. As such, there are two standards to use depending on whether code will be outward facing– to be used by students– or inward facing.

For outward ('public') classes, functions, and variables:

- Priotitize short, easy to spell names– ideally one word
- Limit the number of function/constructor arguments
- Make lines of code read like English; i.e `until(button.pressed)`
- Remove complexities whenever possible

For inward classes, functions, and variables you can follow standard Python form for the most part:

- `lower_snake_case` (even for classes)
- Each class gets its own file
- Limit lines to 80 characters when possible
- Use type hints for both function arguments and return types
- For functions and variables that are not meant to public, prefix them with `__` to prevent them from being interfered with.

Lastly, there are some rules required for compatability with CIRCUITPython and BAP-CAT (described later):

- Not all Python libraries are available on the PIs; always check to make sure your code still runs after adding an import. We might have to design implentations of some things ourselves.

2 BAPCAT

3 CONTENT

3.1 CLASSES

3.1.1 button

```
button(pin)
```

Represents a button, either one of the two mounted to the board or one attached later. Requires only a port to be constructed, and can be built from either one of the `BUTTON_PIN`s or one of the `GROVE_PIN`s.

```
button.pressed()
```

Returns `true` if the button is pressed, and `false` otherwise.

3.1.2 distance_sensor

3.1.3 large_motor

```
large_motor(pin_set, direction = 1)
```

Represents a DC motor mounted on one of the 2 DC motor ports. Requires only a `pin_set` in `DC_PIN`s to be constructed, which limits the number of `large_motor`s to 2.

`direction` is an optional variable that changes the direction of the motor, either positive or negative.

```
large_motor.spin(speed, time = None)
```

Takes a `speed` in the range $[-100, 100]$ (all values outside the range are constrained) and runs the motor at that speed. Optionally, a `time` can be passed in seconds and the motor will only spin for the allotted time before stopping.

```
large_motor.stop()
```

Equivalent to `large_motor.spin(0)`.

3.1.4 small_motor

```
small_motor(pin_set, direction = 1)
```

Represents a servo motor mounted on one of the 4 servo motor ports or one of the grove ports. Requires only a `pin_set` to be constructed.

`direction` is an optional variable that changes the direction of the motor, either positive or negative.

```
small_motor.spin(speed, time = None)
```

Takes a `speed` in the range $[-100, 100]$ (all values outside the range are constrained) and runs the motor at that speed. Optionally, a `time` can be passed in seconds and the motor will only spin for the allotted time before stopping.

```
small_motor.stop()
```

Equivalent to `small_motor.spin(0)`.

3.1.5 wagon

3.2 FUNCTIONS

3.2.1 pause

```
pause(time = None)
```

If a `time` is passed, waits for the allotted `time`. Otherwise, waits for 0.005 seconds.

3.2.2 until

```
until(condition)
```

Pauses the program until the passed `condition` is satisfied.

3.3 CONSTANTS

3.3.1 BUTTON_PIN

3.3.2 DC_PIN

3.3.3 FRQ

3.3.4 GROVE_PIN

3.3.5 reversed

3.3.6 SERVO_PIN