Getting started with the Pi-puck

Prerequisites

- Epuck2 has to be flashed with custom firmware available here. (refer to this wiki 3.2)
- The SD-card of the Pi-extension has been flashed with the custom OS using the Raspberry Pi Imager: Unpack the *tar.gz* file of the OS. In the Raspberry Pi Imager, choose Raspberry Pi Zero 2 W as the Raspberry Pi you want to write to, the *.img* file inside the unpacked *tar.gz* and the micro-SD of the Raspberry Pi as the medium to write to.
- The Pi-extension is mounted on the Epuck2 and the selector is set to *A*.

Step 1: Connect to the Pi-extension

- Via SSH (recommended):
    - First, you need to connect to our WiFi. We will give you the credentials during the exercise.
    - Then, open a terminal and type *ssh pi@pi-puck<id>* and log in with password *raspberry.*
    - You should be connected now. To end the connection, just type *exit* and press *enter.*
- Via Micro USB:
    - First, connect the extension to your PC via a micro-USB cable in the USB-port in the middle (NOT the Pi itself (highest USB-port) or the E-puck (lowest USB-port))
    - Open a terminal
    - If you have not installed screen, do so with *sudo apt install screen*
    - type *sudo screen /dev/ttyUSB0 115200-8N1* (if this shows a *file not found* error, try other ports like */dev/ttyUSB1* etc.).
    - Wait for a few seconds. In the terminal, now the following line should come up:
      Raspbian GNU/Linux 10 pi-puck40 ttyS0

      pi-puck40 login:

      Log in with the username *pi* and the password *raspberry*.
      Sometimes, screen does not show anything. In this case, try SSH instead.
    - Afterwards, you should see something like the following:
      pi@pi-puck<id>:~$
    - To leave *screen*, press *Ctrl + A + D*

Step 2: Run the test script

- connected to the Pi-puck, type *python3 Pi-puck/e-puck2/e-puck2_test.py* and press *enter*

Step 3: Install Pi-Puck API and MQTT

- To steer the underlying E-puck with the Pi-extension, you need to first install the Pi-puck API on your Pi-extension.
- Connect to your Pi-extension via Step 1.
- Type *git clone* [*https://github.com/genkimiyauchi/pi-puck.git*](https://github.com/genkimiyauchi/pi-puck.git) and press *enter.*
- In the same directory, type *pip3 install pi-puck/python-library* and press *enter.* This installs the Pi-puck API as a Python package that you can use system-wide by including the line *from pipuck.pipuck import PiPuck* in your Python script.
- Run *pip3 install VL53L1X* to install an additional package needed for the Pi-puck API.
- Run *pip3 install paho-mqtt* to install the MQTT package for Python.

Step 4: Set up your Git:

- To transfer the Python script you wrote to the Pi-puck, please use a git repository (in the following called *YourRepo*). This way, it is easy to update the code on your Pi-puck and you have a history and backup of your code.
- Create a new git repository or use an existing one. For this, see the induction document of week 1.
- Copy the Python script from moodle (LINK) to *YourRepo* and push it.

Step 5: Get acquainted with MQTT

- MQTT is the protocol we use to publish global position information via WiFi so that the Pi-pucks can access their localization information. For this, we provide a server/broker with the IP address *192.168.178.56*, which publishes a dictionary consisting of the id, position and angle of every robot to the topic *robot_pos/all*.
- To subscribe to an MQTT-topic, we use the paho-mqtt package in Python.
- Initialize the client using *client = mqtt.Client()*.
- For the event *on_connect*, define a Callback function that subscribes to the published topic.
- For the event *on_message*, define a Callback function that decodes the message and stores the data in a global variable to be able to use it later on.

Step 6: Define Pi-puck behaviour

- To initialize your Pi-puck, use *pipuck = PiPuck(epuck_version=2)*
- To steer the underlying epuck, have a look at the functions the Pipuck package provides (e.g., *pipuck.epuck.set_motor_speeds(int, int)*)
- Now you are ready to code the behaviour of the Pipuck according to the tasks.

Step 7: Pull and run the code on the Pi-puck

- First, push your code to *YourRepo*.
- Connect to the Pi-puck (see Step 1).
- Clone *YourRepo* to the Pi-puck.
- Run the script using *python3 client.py*

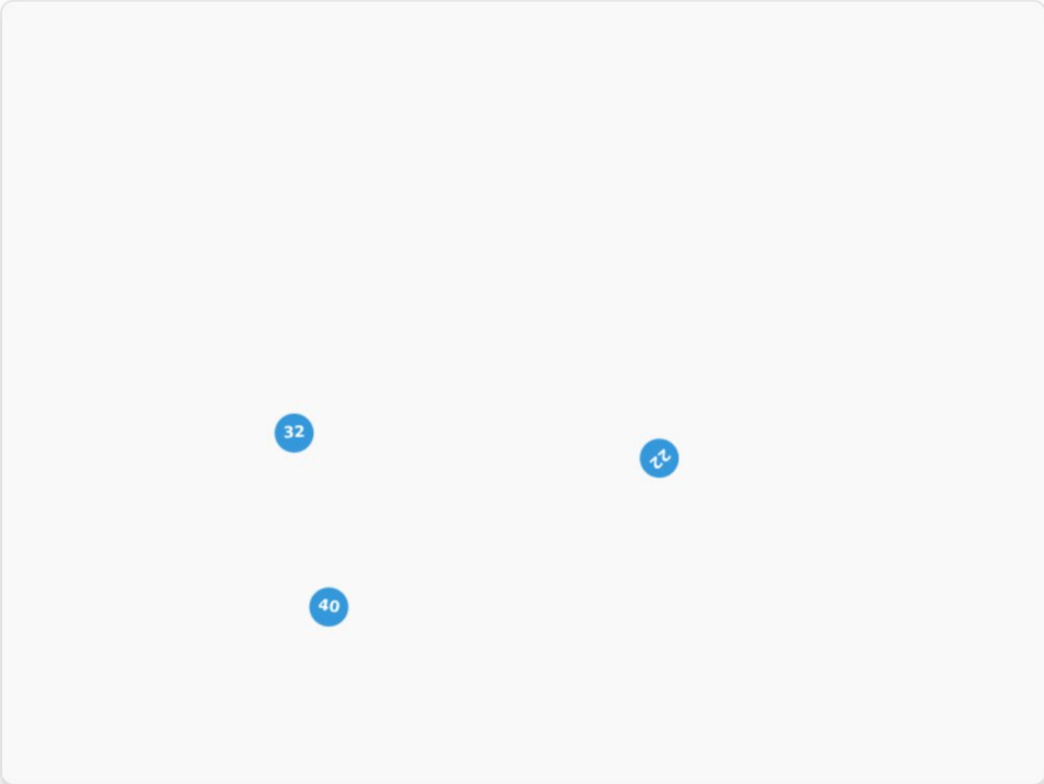
- Every time you update your script, you have to push it to *YourRepo* from your laptop, then pull it on the Pi-puck before executing it.

Step 8: Monitor your Pi-puck in the arena

- In your browser, type 192.168.178.56:3000 and press enter. This will show you the RCPS Tracking Dashboard. Here you can see a bird view of the robots in the arena.
- Below, click on all robots you want to monitor.
- For the last-clicked robot, its ID, position and angle is shown.
- By ticking Show Robot Trails, you will be able to see the trails of visible robots of the last 15 seconds.

RCPS Tracking Dashboard

Show Robot Trails



Robot Info

Robot ID: 22
Position: [0.241, 0.167]
Angle: 138.0°
Visible: Yes

22 22

32 32

40 40

Task 1: Random Walk

Objective:
Get your robot moving autonomously in a basic, non-deterministic pattern.
Instructions:

- Program your Pi-puck to perform a random walk within the workspace.
- The robot should change direction at random intervals and avoid leaving the defined area.
- Use the overhead camera system to periodically query your robot's own position via MQTT.
- Bonus: Implement collision avoidance with static objects using available sensors or position data.

Task 2: Talk to Close Robots

Objective:
Enable your robot to discover and communicate with nearby robots.
Instructions:

- Each robot should subscribe to its own MQTT topic (e.g., robot/<id>).
- Extend your random walk so that every few seconds, the robot queries the positions of all other robots using MQTT.
- If another robot is within a defined communication radius (e.g., 50 cm), send a message (e.g., "Hello") to its topic.
- Upon receiving a message, the recipient robot should log it and blink an LED or display a notification (if applicable).

Task 3: Catch Me If You Can (find in groups of 2-4 students)

Objective:
Create a tag game where one robot (the **runner**) avoids others (the **chasers**).
Instructions:

- Choose one robot to act as the runner and the rest as chasers.
- The runner performs a modified random walk and actively avoids the closest chaser (based on position data).
- Chasers coordinate (via MQTT) to share runner location data and try to intercept it.
- A chaser "catches" the runner by getting within a certain distance (e.g., 20 cm). On a successful catch:
- The chaser sends a "caught" message.
- Roles may optionally switch (runner becomes chaser).

- Bonus: Implement basic coordination between chasers to trap the runner (e.g., encircle it).