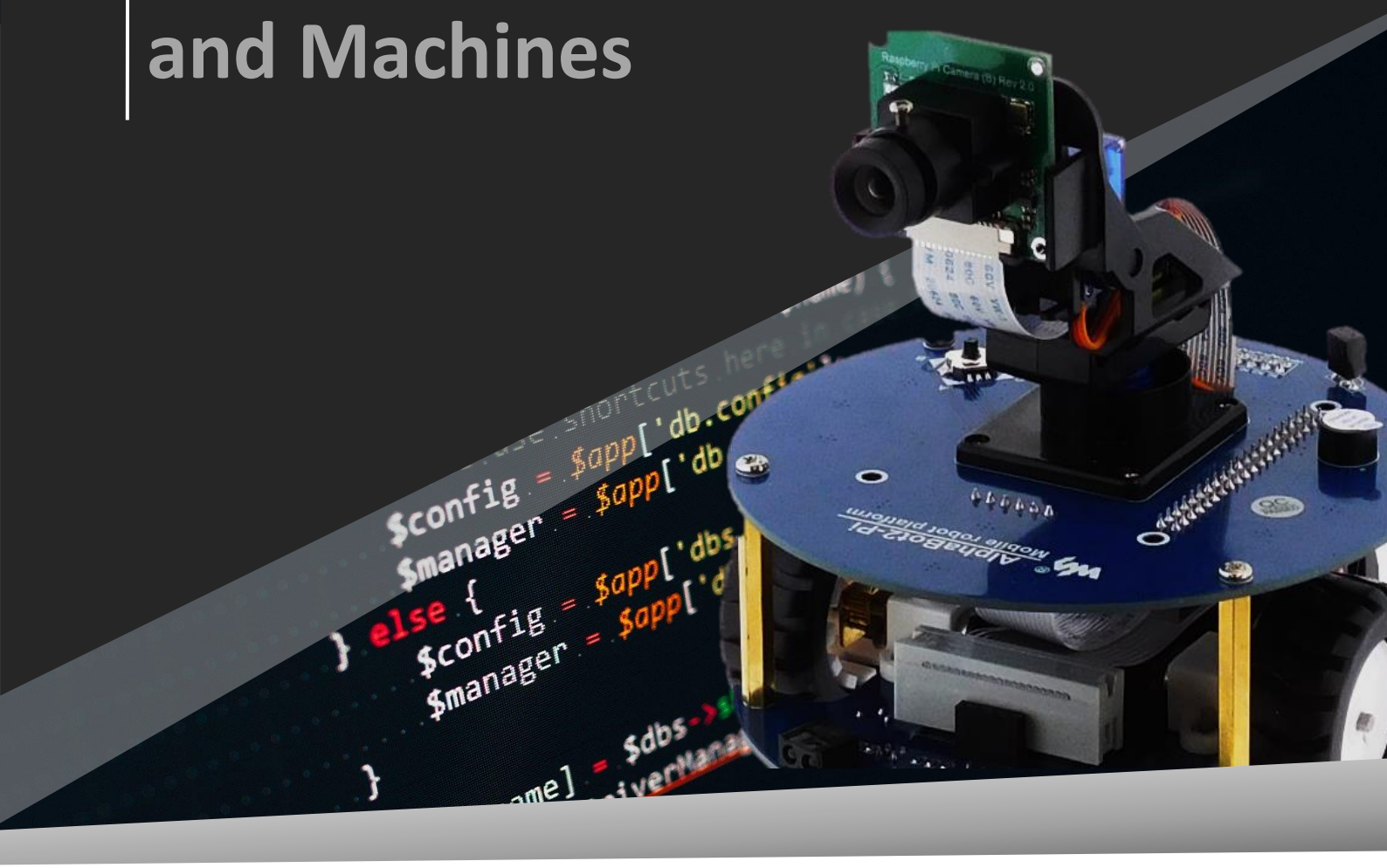
Softwares for Robotics and Machines



Pitch

A three-game entertainment robot

Will you defeat it?

Julien AMADEI Lucas BERNARD Louis GUIGON

ABSTRACT

In this report, we will describe how our robot (Pitch) was designed, explaining the logical reasoning, interesting code snippets, and design philosophy.

Table of Contents

Introduction – What is Pitch?

Pitch is meant to be an "entertainment robot", providing the user with games from a set of three. These include:

1) Rock, Paper, Scissors
2) Guess the color
3) ~~Simon~~ Pitch Says

Used components & software

The robot uses most of the embedded hardware already present on a vanilla Alphabot2 robot to provide entertainment.

This includes:

- DC Motors for moving around
- A Color Camera (a Logitech C270) which is pannable and tiltable
- A Buzzer for vibration
- A set of four RGB LEDs
- One IR Remote for remote control

Other sensors are also present and functional (fully coded), but unused in this project as of now:

- IR Sensors for Object-proximity sensing
- A set of five IR Line Sensors for Line Tracking

Regarding software, we will be using Ubuntu Server 20.04, along with *desktopify* (1)**,** which allows us to get a Desktop interface. We also will be taking cues from the official wiki's *code examples* (2).

Pitch's version of ROS is Noetic (3). However, using Ubuntu on Linux lead to a slew of errors, which we fixed using wikis and threads (4) (5) (6) (7) (8)

The use of a remote desktop (9) allows us to use the robot more easily and use terminals to check for sensor data.

The principle/main game loop

The principle (main game loop) is simple and does not vary outside of the selected game (1/2/3). It is comprised of four Steps:

1) Setup
2) Query (Game choice)
3) Game Loop
   a. Rock, Papers, Scissors
   b. Guess the Color
   c. Pitch Says
4) Results

Each step will be described in the next chapter, along with code snippets and illustrations for a better understanding.

All codes mentioned are open sourced on our GitHub repository
https://github.com/JulienAmadei/srm-pitch

Or just use this !

- A user guide is also present in the last page of this report, as an annex.

Main Loop Description

Step 1 - Setup

From the User's point of view

For the proper experience, the user should stand next to a table. It is also possible to play while sitting down. However, in this scenario, the user should be ready to adapt the camera angle manually.

The robot should be put on the same table, which allows it to stand at the correct height relative to the user.

- Once the startup is complete, the robot will look around to figure out where the user is relative to its current position using its camera. If it is too far from them, it will move towards them, to better understand what the user wants.
- If there is no user, Pitch will move around with a 360-degree sequence trying to look for someone.

The code

Main principle

The robot will be initialized with the global launch file to start all the necessary nodes using ROS. Once everything is initialized, the camera and its servos will try to find a user. This step allows us to have a 100-degree range to look for a user before turning the robot on itself to reach other angles.

<u>This method is implemented as follow, and described in Figure 1:</u>

The servo begins at -50 degrees and ask the camera to find a player. Should the camera find no user in its range, the camera will pan with a 25-degree increment, and the loop continues.

As the panning angle reaches 50-degrees without any user found, the robot will turn left to start another cycle. Such a loop will repeat until a user is found.

A verification loop is implemented to avoid camera errors and artefacts: we must validate 5 times that a face is detected before concluding that we have found a user.

Afterwards, the robot will ask the user if they are willing to play. If the user does a "thumbs up" motion to accept, the robot will continue and ask for a game. Otherwise, the robot will act sad. In this instance, a verification loop is also implemented to be sure of a correct choice: the camera must validate 3 times that a thumbs reaction is detected.

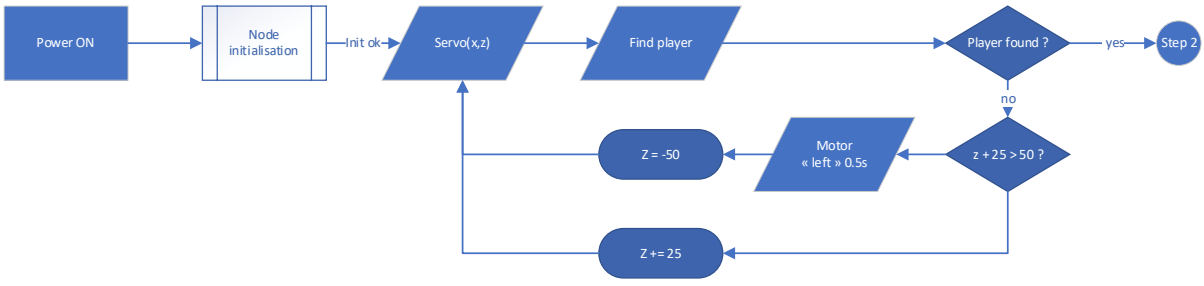


Figure 1 - Robot Initialization and User-Finding logic diagram

Dependencies

As previously stated, the robot's first task uses both the camera and the motors, as it must move around to put the user in the camera's range of view.

Concerning the computer vision aspect, a face recognition code was implemented in a ROS server, ***camera_find_player_server*** (10). It relies on Haar Cascade recognition *(An Object Detection Algorithm used to identify faces in frames)* and contains a function to compute the distance to the user to move to an appropriate distance to play games.

We decided to work with services for the camera because we know that computer vision's algorithms can be quite demanding in resources. Therefore, by using services, we work on images only when asked. Furthermore, the use of a "Service Switcher" allows us to get a specific information out of an image and reduce the complexity of image computing.

Haar Cascades are an inherent part of OpenCV and a very useful library for object recognition. It works on edge detection on data divided between positive and negative images. Positive images must contain the object to detect and negative images must have objects that can be seen in the environment of the object we want to detect.

The distance finder algorithm takes the median value for the width of a face and with the focal length of the camera (computed from a set of images) it can estimate the distance of the user. It might not be proved to be accurate if the user has a finer or wider face than the norm, but we only need a rough estimation to be at a certain distance to the user.

What Pitch "sees" when running the ***camera_find_player_server*** can be seen in Figure 2.

Figure 2 - camera_find_player_server output

Throughout the different camera nodes, the function ***auto_adjust_brightness_and_contrast*** (11) was used. It allows the image to be more constant when the lightening conditions change, as seen in Figure 3.

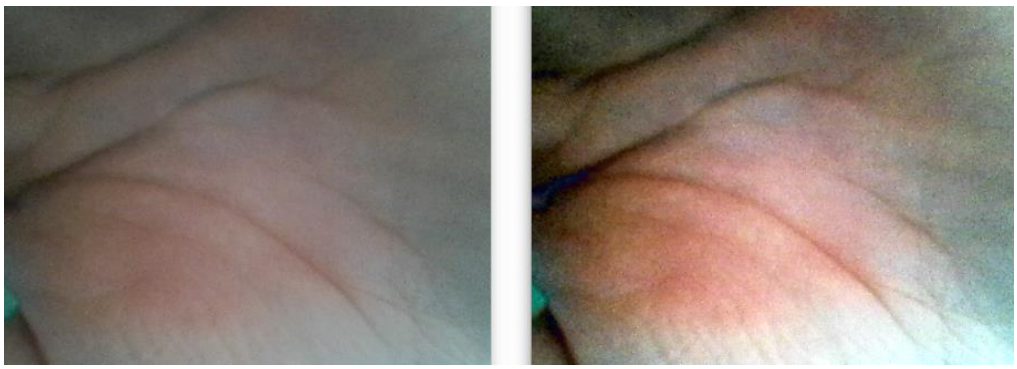


Figure 3 - Left: raw image from camera | Right: adjusted image

To move the servos and the motors, action servers are used:

- For servos, this allows us to send a position goal with the possibility to have position feedback. They are controlled using a goal of *(panAngle, tiltAngle)* with values in degrees, and moved by increments achieved through the same method seen in (2).
- For DC Motors, we are unable to obtain any position feedback easily. Therefore, we must estimate the motion in seconds of execution. Another way would have been to control the motion with computer vision, but after some testing, we were unable to create a code that could produce a satisfactory result using only a Raspberry Pi 4b's resources.
  The motion is achieved through the same method as in (2).

Step 2 – Query (Game Choice)

From the User's point of view

Once the robot found the user, it will prompt them to play by buzzing and lighting up in green.

Figure 4 - Pitch's IR Remote

- The user can **accept** a game by doing a **thumbs up**, which will cause the robot to buzz, light up in white and wait for the game choice, which is dictated using the IR remote seen in Figure 4; each game has an ID linked to a remote button.
  - ***1 → Rock, Papers, Scissors***
  - ***2 → Wheel Game***
  - ***3 → Simon Says***

  *Pressing **0** results in the **cancelation** of the game choice.*
- The user can **decline** by doing a **thumbs down**, which will cause Pitch to look up, light up in red, buzz and back away.

The code

Main principle

After initialization, the robot will look for the user's thumb to try and figure out what their reaction is.

Once it finds a thumbs up, it will begin listening to the IR remote talker, to find out what buttons the user pressed, and launch a game logic from there.

The logic behind these principles can be seen in Figure 5.

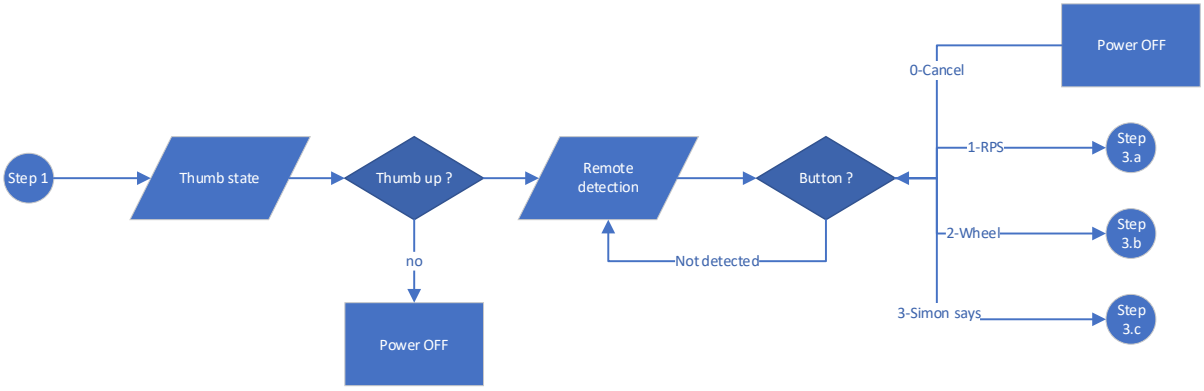


Figure 5 - Game Query and Choice logic diagram

Dependencies

To act as a transition between the different camera servers and the main, ***camera_switch*** was implemented as a server. It takes in argument a string of the object that needs to be detected ("hand", "face", etc.) and returns the appropriate information depending on the input.

For this query step, we use the ***camera_finger_counter_server*** (12) ROS server to detect the thumb orientation.

This node uses the Cvzone (12) and Mediapipe (13) libraries to apply a skeleton hand on the user's actual hand. With this new structure, we can detect whether the thumb is up or down and count the number of fingers held up. This functionality will be described in further details in the **Rock, Paper, Scissors** section.

Figure 6 is what Pitch "sees" while running ***camera_finger_counter_server***:

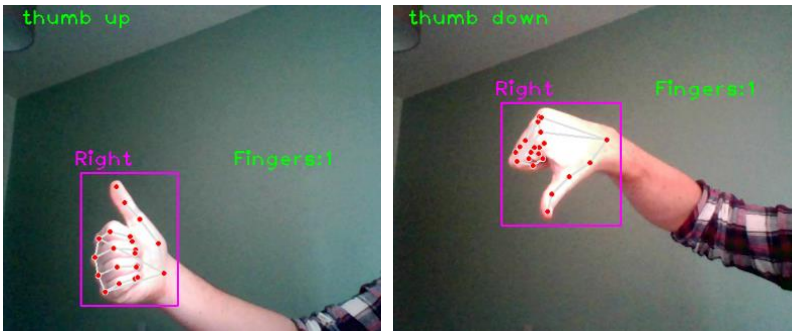


Figure 6 - camera_finger_counter_server output

We also listen to the messages set to the ***remote_input*** topic by the ***remote_talker*** node of our ***ir_remote*** package. While this means that the remote inputs are constantly being "listened" and "transmitted", this also hasn't proven to be impactful on performance. Furthermore, this allows for a quick change of mind, without having to call a service again.

We also use the LEDs and Buzzer for visual and sound feedback. These are respectively handled using the ***buzzer_server*** and ***LED_server*** ROS servers to interact with the Raspberry's GPIO pins. The code is straightforward and already present in (2). Our version just adds a wrapper for ROS, to be able to use it as a Service, and do it concurrently with other tasks.

Step 3 – Games

a. Rock, Paper, Scissors

From the User's point of view

As the user chooses to play Rock, Papers, Scissors, the robot will light up in Purple to show the selected game mode and vibrate to have feedback and tell the user to make a move.

At the same time, Pitch will decide on a random move, and the LEDs will light up in a static color, indicating Pitch's "choice". The choices are described in Table 1.

Blue	**Rock**
Green	**Paper**
Red	**Scissors**

Table 1 - Rock, Paper, Scissors moves and their relative colors

The robot then assesses the user's move and find out who won, with the classic rules presented in Figure 7.

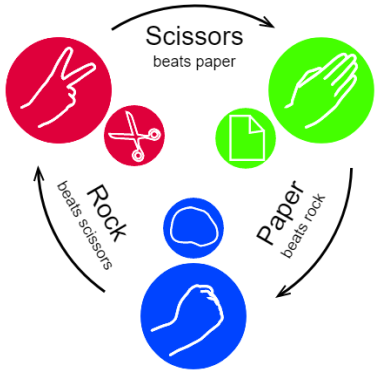


Figure 7 - Rock, Paper, Scissors Game Rules

Note: In the case of a tie, the game will restart.

The code

Main Principle

This game is implemented as follows, and described in Figure 8Figure 1:

As the robot blinks and vibrates, a random number generator is used to generate a random move choice. The robot then uses its camera to figure out the user's move, using computer vision. For better handling (as the Raspberry Pi has sometimes trouble running the Computer Vision software), we decided to allow for errors in the finger numbers. Therefore:

- If 0 or 1 fingers are held up, Pitch will acknowledge the move as **Rock**.
- If 2 or 3 fingers are held up, Pitch will acknowledge the move as **Scissors**.
- If 4 or 5 fingers are held up, Pitch will acknowledge the move as **Paper**.

Once both players' moves are figured out, they are interpreted by a game logic manager, handling win, losses, and ties, and passing them to the robot's reaction logic (described in Step 4). In the case of a tie, the game will restart.

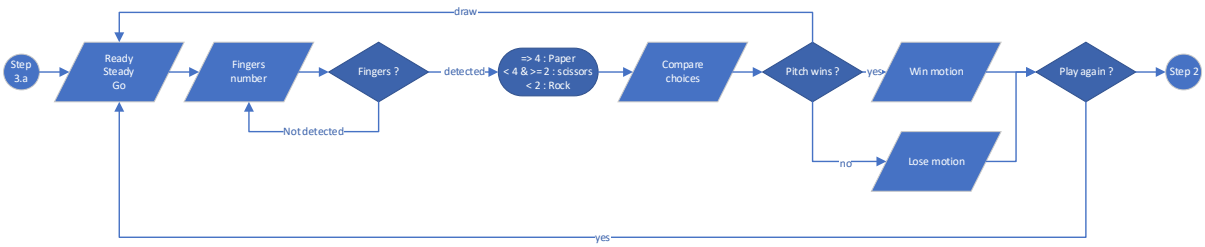


Figure 8 – Rock, Paper, Scissors game logic diagram

Dependencies

For this game, we use once again the ***camera_finger_counter_server*** ROS server to detect the number of fingers held up.

Figure 9 describes what Pitch "sees" while running ***camera_finger_counter_server***:

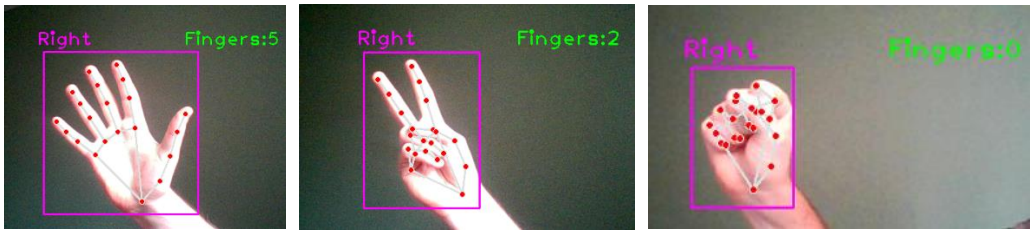


Figure 9 – Rock, Paper, Scissors interpreted by Pitch

Once again, visual and sound feedback is once again handled using ***buzzer_server*** and ***LED_server***.

b. Spin the Wheel

Figure 10 - Pitch's RGB Wheel

From the User's point of view

Pitch will light up in Turquoise to show the selected game mode, and the user will have to spin a custom-made tricolor RGB Wheel, as shown in Figure 10.

As the wheel spins, Pitch will vibrate and decide of a random color, either red, green, or blue.

The robot will then use its camera to assess the color once the wheel has stopped spinning to find out if it was right or not.

The code

Main Principle

The game is implemented as follows, and described in Figure 11Figure 1:

As the robot blinks and vibrates, a random number generator is used to generate a random color choice. The robot then lights up in said color and uses its camera to figure out the color the user is showing using computer vision.

Once the robot guessed and figured out the wheel result, they are compared, and passed to the robot's reaction logic (described in Step 4).

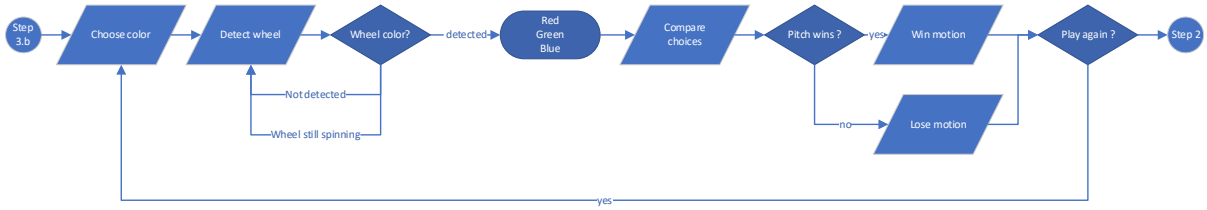


Figure 11 – Wheel game logic diagram

Dependencies

For this game, we use the ***wheel_server***, the node calls the ***color_selection_server*** with special requirements: it will detect the three different colors of the wheel, we calculated beforehand the values in input, and compute their barycenter. Back in the ***wheel_server***, it will compare the heights of each part and find which one is the lowest. This color will then be compared to the Pitch order.

Once again, visual and sound feedback is once again handled using ***buzzer_server*** and ***LED_server***.

c. Pitch Says

From the User's point of view

Pitch will light up in Yellow to show the selected game mode, the robot will vibrate and indicate a fixed color using its LEDs.

The player must show him an object of the stated color during a time limit, indicated by the LED countdown, and Pitch will assess the result.

The code

Main Principle

The game is implemented as follows, and described in Figure 12Figure 11Figure 1:
As the robot blinks and vibrates, a random number generator is used to generate a random color choice. The robot then displays the color to find using its LEDs uses its camera to figure out the result color, using computer vision.

Once time runs out, or the user shows the guessed color, they are compared, and passed to the robot's reaction logic (described in Step 4).

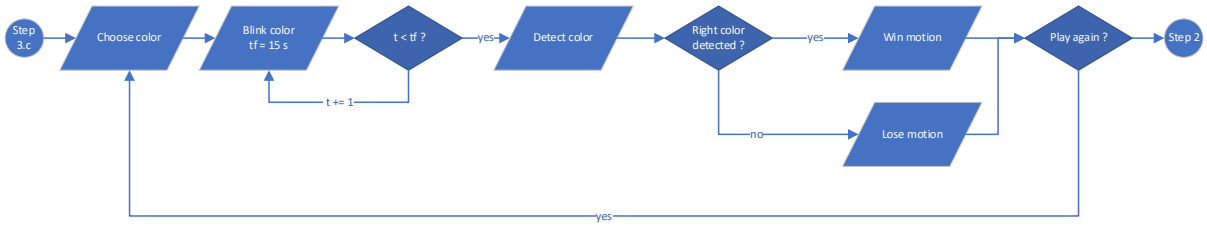


Figure 12 – Pitch Says game logic diagram

Dependencies

For this game, we use the ***color_selection_server*** (14), the node will apply a filter to the frame to see if the color is part of it, the input is then a string with the color name given by Pitch and the output is a Boolean that certifies the occurrence. In background, there is a dictionary with associations to give a HSV mask to each color, as seen in Figure 13.

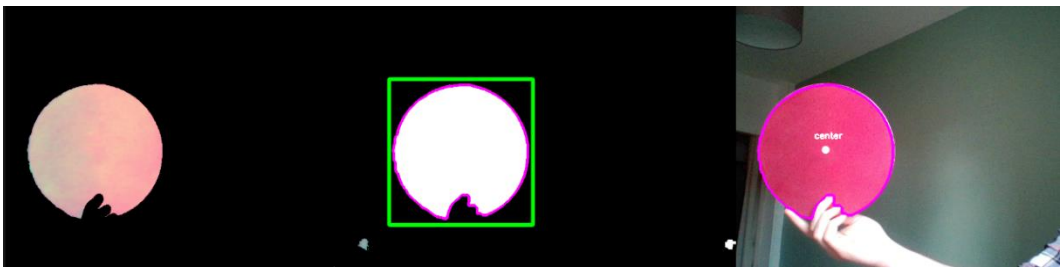


Figure 13 - Steps for the color_selection: color filter, contour it and get the result

Step 4 – Results

From the User's point of view

Pitch's behavior depends on whether it won on lost.

- Should Pitch win, it will nod its head, blink, and vibrate.
- If Pitch loses, it will shake its head and light up in red.

It will then ask the user to play another round.

- If the user **agrees** to another game (thumbs up), the game loop continues.
- If the user **declines** (thumbs down), the loop goes back to step 2, and another game can be selected.

The code

Main principle

To avoid redundancy, the main principle will only be illustrated using the simple diagram seen in Figure 14.

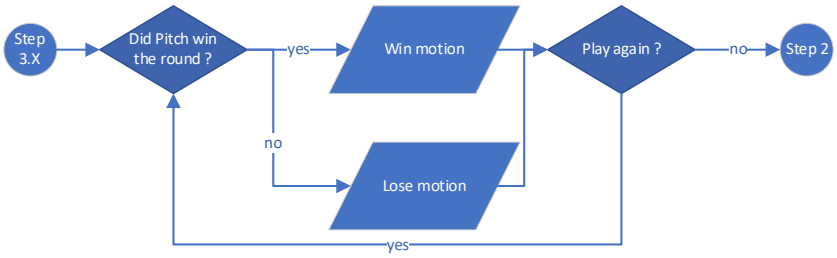


Figure 14 - Pitch game reaction logic diagram

Dependencies

For recognizing the user input using the camera, we are once again using the ***camera_finger_counter_server***, while visual and sound feedback is handled using ***buzzer_server*** and ***LED_server***.

To make the robot "dance", we use the servos and motors action servers described previously.

Overview of the whole structure

Simplified graph

The graph shown in Figure 15 shows our strategy to build nodes in this project.

The links between nodes shows the global communication system and summarize the exchanged information. The master and topics are not represented here for a better comprehension. The real graph is shown in the next section.

We thought for a future work on the project, so we prepared it already. For instance, the nodes for Infrared servoing are already coded and the servos and motors are built as actions to have feedback. All things are ready for a position control.

Then, we still have a huge ***Pitch_client*** node. It is due to a lack of time in this project. All games, detections and choice loops could be separated in different nodes, but we did not have the time to make it perfect.

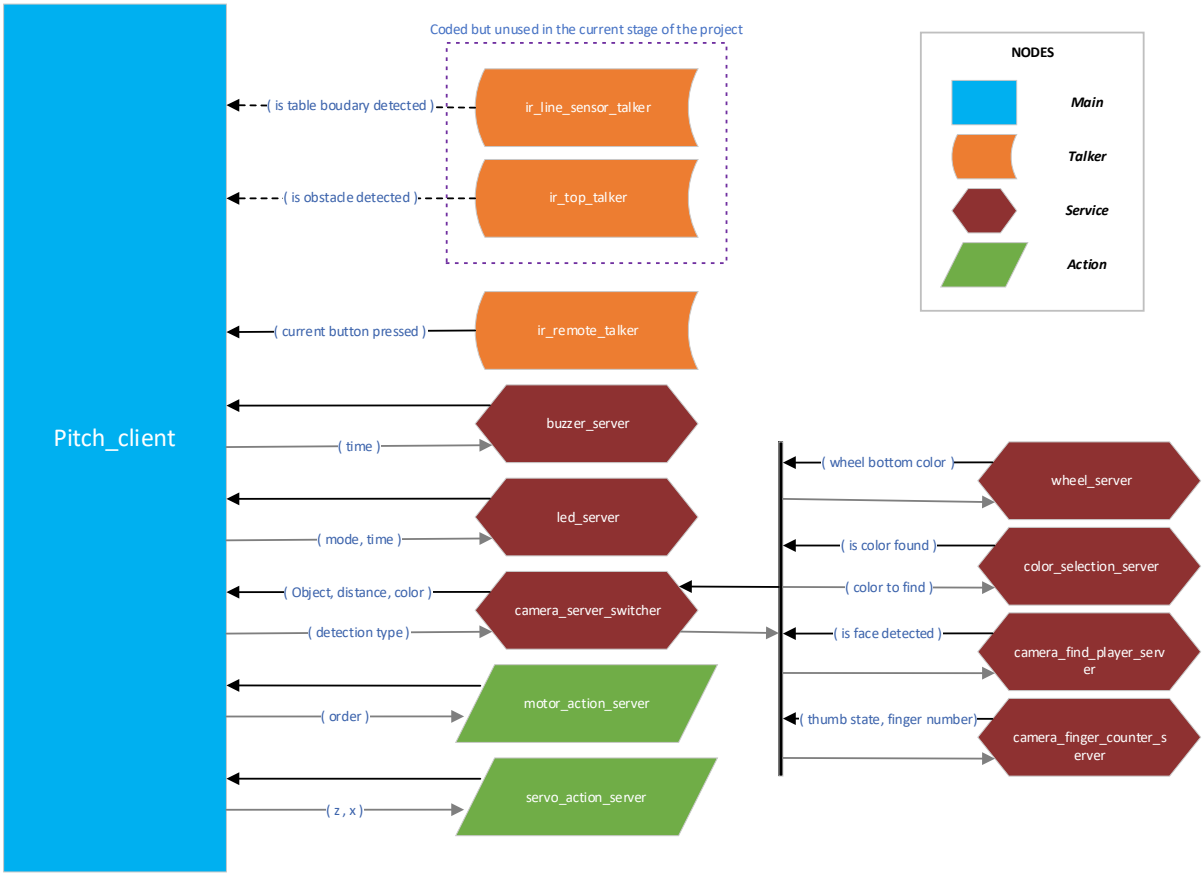


Figure 15 – Pitch's main logic diagram

rqt graph

ROS has its own representation of the nodes, called the rqt_graph (see Figure 15). By calling it in a terminal with an instance of roscore running, it shows all connections between current nodes.

In our case, it is efficient to see the current connections. However, as a lot of services are used, it can be difficult to differentiate them because they are all linked to the master (/rosout).

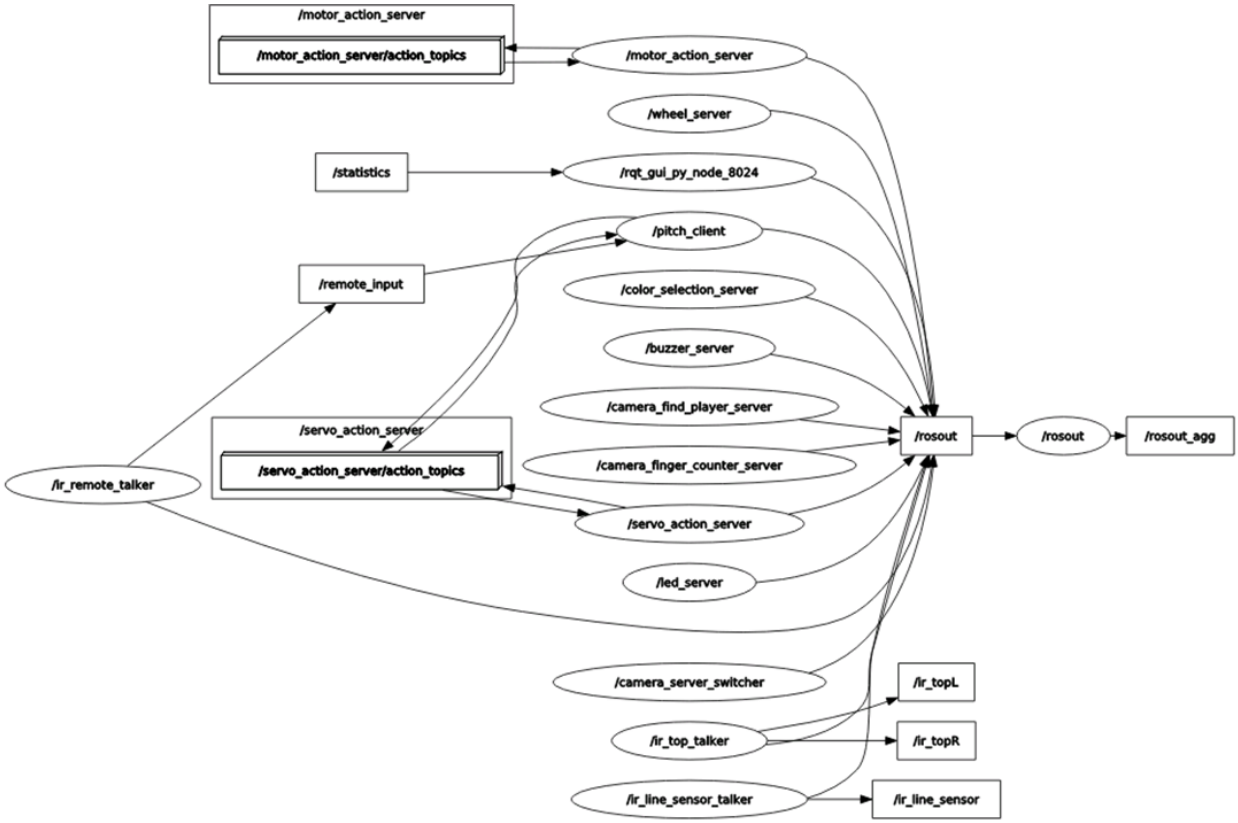


Figure 16 - rqt_graph diagram of Pitch while in use

Conclusion

This project was limited by the time given. We still have a lot of games we would have liked to add (and made small PoCs of), such as “red light, green light”. Furthermore, we spent a lot of time to build a stable OS, which was not the main point of this class or project. The main issue with Ubuntu is user rights management, mostly because with the default configuration the code must be run as administrator, and ROS is not one by default. In other words, we had to define all roles for GPIO, SPi… to make it work with ROS. This is easily fixed by using Debian (Raspberry Pi’s default OS), but doing so comes at a cost of compatibility for most of our libraries.

Another limiting aspect is the ressource allowed. We were always looking at reducing code complexity and maximizing efficiency to keep the Raspberry Pi alive when running demanding Computer Vision operations. It’s a point that we do not usually take into account, but has proven to be an interesting challenge. An easier option would have been to get an NVIDIA Jetson Nano, made for IA and Computer Vision Tasks, or an Intel Realsense Camera (although they have recently been put off market)

Finally, using ROS along with Git was useful for efficient teamworking, as we could work together on the project without waiting for the robot to be available, or for unrelated files to be updated. Another advantage is that the code is referenced in a GitHub repository and can be evolved in the future if it need be, as long as proper credit is given.

ROS has also proven to be a strong way to code a robot. From adding nodes to a current project to finding errors when running the project as a whole, it helps in creating an efficient work environment. Most of our nodes and packages are not perfect (comments are lacking, not every package has a launch file, and package.xml files are not fully filled), but the main objectives we set are competed in time. Therefore, we believe to have understood how to choose between a Talker, a Service or an Action Server for specific actuator or sensor communication.

References

1. **Wimpress, Martin.** Desktopify - Convert Ubuntu Server for Raspberry Pi to a Desktop. [Online] 12 3, 2021. https://github.com/wimpysworld/desktopify.
2. **Alphabot2.** Alphabot2 - Official Wiki. [Online] 2017. https://www.waveshare.com/wiki/AlphaBot2-Pi.
3. **ROS.** ROS WIKI. [Online] https://wiki.ros.org/.
4. **webhostinggeeks.** How to Add User to root Group on CentOS 5/CentOS 6. [Online] https://webhostinggeeks.com/howto/how-to-add-user-to-root-group-on-centos-5-7/.
5. **jgarff.** rpi_ws281x - Fixes for arm64 support. *GitHub.* [Online] https://github.com/jgarff/rpi_ws281x/pull/316/files.
6. **permissions, Launch node with root.** answers.ros.org. [Online] 2014. https://answers.ros.org/question/165246/launch-node-with-root-permissions/.
7. **davidcorbin.** RuntimeError: Not running on a RPi! with Ubuntu for Raspberry Pi. [Online] 2020. https://github.com/gpiozero/gpiozero/issues/837.
8. **scottlawsonbc.** Big Ol' Christmas Update. [Online] 2020. https://github.com/scottlawsonbc/audio-reactive-led-strip/pull/101.
9. **Neutrinolabs.** XRDP GitHub. [Online] https://github.com/neutrinolabs/xrdp.
10. **GeeksForGeeks.** Distance Finding using OpenCV. [Online] https://www.geeksforgeeks.org/realtime-distance-estimation-using-opencv-python/.
11. **OpenCV, Autoadjust brightness on.** [Online] https://coderedirect.com/questions/121557/automatic-contrast-and-brightness-adjustment-of-a-color-photo-of-a-sheet-of-pape.
12. **CVZone.** CV Zone Official GitHub. [Online] 2021. https://github.com/cvzone.
13. **Google.** Mediapipe Official GitHub. [Online] 2022. https://github.com/google/mediapipe.
14. **Projects, OpenCV Python Tutorials and.** Color, Edge and Object detection tutorials. [Online] 2021. https://github.com/murtazahassan/OpenCV-Python-Tutorials-and-Projects.
15. **Make a ROS Camera Publisher/Subscriber. [Online] Automatic Addison. https://automaticaddison.com.**

Figures

Rock, Paper, Scissors

(Button 1)

I'll vibrate and it'll be your time to make a move!

Meanwhile, I'll decide on a move myself and light up accordingly!

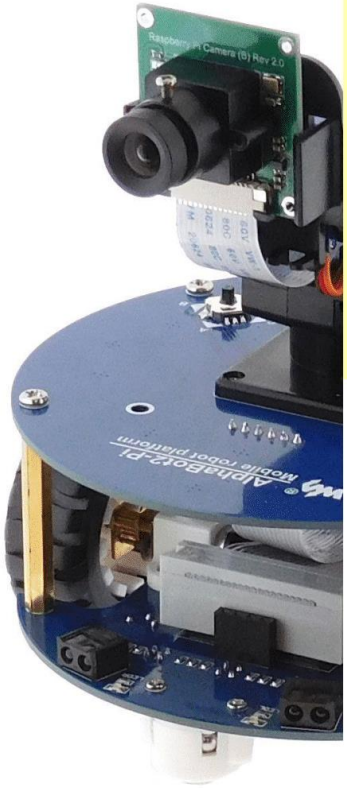
Spin the Wheel

(Button 2)

First, you'll have to spin my custom-made Wheel.

As it spins, I'll vibrate and decide on a random color.

Once the wheel stops spinning, we'll see how good I am at predicting the future!



~~Simon~~ Pitch Says

(Button 3)

I'll vibrate and indicate a fixed color using my LEDs.

Try and bring me an object of the same color as soon as possible! I'll set a time limit to test your speed!

(Using the wheel is prohibited! Please don't cheat, I'm still new at this !)

Hello there !

To play with me, just let me find you !
Once I can see you, just give me a thumbs-up, use the remote buttons and we can play !

AMADEI Julien
•••
BERNARD Lucas
•••
GUIGON Louis
•••
Softwares for Robotics and Machines
•••
2022

PITCH

USER GUIDE

github.com/JulienAmadei/srm-pitch

ROS