

MiRo - a learning pet robot

CS39440 Major Project Report

Author: Aleksandra Madej (alm82@aber.ac.uk)

Supervisor: Dr Patricia Shaw (phs@aber.ac.uk)

19th February 2019

Version: 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Science
and Artificial Intelligence (GG4R)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Aleksandra Madej

Date: 02.05.2019

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Aleksandra Madej

Date: 02.05.2019

Acknowledgements

I would like to thank my project supervisor, Patricia Shaw, for providing support and guidance during my work on the project.

I am also grateful to Teresa Walczak, my high school mathematics teacher who inspired me to study a Computer Science and Robotics related degree.

Abstract

In recent years robotic toys have been becoming more and more popular. Although they are currently mostly used for entertainment, in the future robot pets could provide a good alternative for a real animal companion.

Thanks to their low maintenance needs this kind of robots could become a useful tool for teaching young children how to interact and take care of animals, but also act as a companion for those who are no longer able to look after real pets.

As the technology advances, resemblance of robotic pets to real animals increases. However there is an important aspect in their design that is often neglected - animal's ability to learn. Most existing robot pets offer a set of tricks that can be performed with use of appropriate commands. However these are mainly predefined behaviours that cannot be changed or modified.

In the real pet and human relationship, teaching helps to increase the bond between them and disciplines the behaviour of the pet to adjust it to the owner's personal preferences and needs.

To satisfy those needs, a trick learning system will be developed as the technical part of the project. The robot will be able to learn both commands and ways to respond to them in order to simulate the interaction with the pet better. Additionally a simple monitoring system will be developed to allow easier control over the learning process.

Contents

1	Background & Objectives	1
1.1	Background	1
1.1.1	Motivation	1
1.1.2	Background preparation	1
1.1.3	Similar systems	2
1.2	Analysis	2
1.2.1	Objectives	2
1.2.2	Emotions	3
1.2.3	Command detection method	4
1.2.4	Learning system	7
1.2.5	Other Choices	7
1.2.6	Deliverables	8
1.3	Process	8
2	Design, Implementation and Testing	9
2.1	Overall Approach	9
2.1.1	Design	9
2.1.2	Implementation	10
2.1.3	Testing	10
2.2	Actions	12
2.2.1	Design	12
2.2.2	Implementation	13
2.2.3	Testing	13
2.3	Emotion System	15
2.3.1	Design	15
2.3.2	Implementation	16
2.3.3	Testing	16
2.4	Command Detection	17
2.4.1	Design	17
2.4.2	Implementation	19
2.4.3	Testing	20
2.5	Learning System	22
2.5.1	Design	22
2.5.2	Implementation	23
2.5.3	Testing	24
2.6	Sequence Building	25
2.6.1	Design	25
2.6.2	Implementation	26
2.6.3	Testing	26
2.7	Monitoring GUI	27
2.7.1	Design	27
2.7.2	Implementation	28
2.7.3	Testing	28
2.8	Final Design	31

3	Evaluation	32
3.1	Analysis and research	32
3.2	Design, Implementation and Testing	33
3.2.1	Design	33
3.2.2	Implementation	33
3.2.3	Testing	34
3.3	Future Work	34
	Annotated Bibliography	35
	Appendices	36
A	Third-Party Code and Libraries	38
B	Ethics Submission	39
C	Shape cards	43

List of Figures

1.1	Example sound data	5
1.2	"Turn left" command- first recording	5
1.3	"Turn left" command- second recording	5
1.4	"Move forward" command	6
2.1	Initial class diagram	10
2.2	Output of the command detection	19
2.3	Comparison of input and output	19
2.4	Detection with shape not fully in camera range	20
2.5	Positive feedback to "stop action"	26
2.6	Negative feedback to "stop action"	26
2.7	Initial Design	27
2.8	GUI: Initial State	29
2.9	GUI: New command added	29
2.10	GUI: Command Information	30
2.11	GUI: Command currently detected	30
2.12	Final system design	31

List of Tables

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Motivation

The decision on choosing this project has been influenced by multiple factors. One of them was the opportunity to work with MiRo - an advanced type of robot using variety of different sensors - but also learn many new technologies. Although that meant a large amount of time would be spent getting familiar with techniques and tools not used before, it would also result in valuable experience being gained. Another reason was a high degree of freedom offered by the project which allowed more personalization compared to other available projects.

1.1.2 Background preparation

The initial presentation of MiRo showing its dog-like appearance and use of the emotion system inspired the idea of creating a trick learning software for the robot. However to decide on the system features, further research on its capabilities had to be performed. Available sensors and actuators [1] have been investigated and their quality and limitations have been evaluated. Furthermore features provided by existing emotion system [2] have been explored. Preparation has also included becoming familiar with ROS framework used to communicate with the robot. Concepts of nodes, messages and topics [3] have been studied and choice of available programming languages has been considered. The research has been followed by a series of practical experiments based on example programs provided with the robot. The programs were analyzed and modified to access data from various sensors and alter the robot's behaviour.

Additionally a research on creating a realistic user experience while interacting with companion robots has been conducted. The need of personalization and individualization of pet robot behaviour has been identified to build a natural relationship [4] - the robot should be able to adapt to the user's needs and preferences.

1.1.3 Similar systems

Research on similar systems has been performed to evaluate the project's usefulness but also collect ideas and inspirations for specific system features. Comparable companion pet robot systems include:

- AIBO [5] - robotic pet created by Sony. The robot is equipped with an emotion and desire system dictating its behaviour. It is able to detect and avoid obstacles, react to sounds, as well as perform tricks. AIBO understands a set of voice and visual commands, but can also learn new simple actions. Pushing the pads on its paws while moving them allows it to memorize the movements and repeat it in the future with use of the mobile application [6].
- Genibo QD [7] - robot dog manufactured by Dasarobot of Dasatech. Like AIBO it can navigate around obstacles and show its emotions to the users. The robot understands around 100 voice commands and can be taught dance moves via PC software
- Robopet [8] - robot produced by WowWee. Compared to previous robotic pets this one is less advanced. It is able to wander autonomously and perform tricks activated by the user via the remote control.

All of the robots described above have an ability to perform tricks. However none of them offers a natural way of teaching the robot a new trick. Although AIBO provides a way of teaching sequences of movements, these can only be accessed via mobile application and not as reaction to given command.

1.2 Analysis

1.2.1 Objectives

The research of similar systems has resulted in direction of the project being chosen. A system for learning tricks through interaction would be created for the robot. Overall design of the software has been created and the general objectives for the system have been identified as follows:

Upon launching the program, the robot will know no commands or tricks - it will only be capable of performing simple actions such as "go forward" or "turn right". MiRo will also be able to detect and recognize commands given by user and develop separate sequences of actions for each of them .

When new command is detected it will perform random actions and wait for feedback to learn a new trick. Positive emotions, triggered by user feedback (touching the robot) will increase the chance of the same response being chosen in the future. If MiRo detects command it already knows it will perform set of actions associated with the best user's reaction and proceed with random actions allowing to extend the existing sequence.

Learning rate will also be affected by MiRo's overall mood for increased realism - it will learn new tricks much faster if its happy and refuse to learn anything while annoyed or anxious. This will also mean that it will learn faster the longer the training session is, as the positive feedback required for learning will keep improving its mood.

The project will also involve developing a simple system for monitoring MiRo's learning progress. This will display commands recognized by robot as well as sequences of actions associated with them

Taking above objectives into consideration the system has been divided into following features:

- Actions - responsible for performing simple actions such as turn left or move forward
- Emotion System - used for controlling Miro's emotion and mood
- Command Detection - a feature in charge of detecting, recognizing and responding to commands
- Learning system - a system used to modify chances of performing specific actions in response to specific commands
- Sequence building - a feature allowing to build more complex tricks out of simple actions
- Monitoring system - a graphical interface helping to monitor the learning process

Other considered features were idle behaviour and save and load option, however both of them have been eventually abandoned. Although the idle behaviour would be necessary for a full intelligent robotic pet system it is not an essential part of the learning software. Additionally it could be confused for a part of the trick and affect the understanding of performed actions. Similarly the save and load options would be a useful addition and would ideally be included in the system, however considering the time frame of the project it would be difficult to find additional time to implement it.

1.2.2 Emotions

Initially the use of the existing emotion system was considered as it was an important factor that inspired the direction of the project. After further evaluation however, a conclusion was made that using it could cause confusion while assessing work accomplished during the project. As the provided emotion system alternates the robot's behaviour, emotion based behaviours and actions being part of tricks would be difficult to distinguish. Therefore a new simple system would have to be developed.

1.2.3 Command detection method

Many of initial ideas were rejected after the research on the robots sensors and actuators. Using the cameras for complex computer vision was not an option due to their low resolution and low quality of microphones would not allow for speech recognition. After rejecting ideas unlikely to succeed, options left to consider included: sound pattern recognition, pose detection or shape and colour detection.

Sound pattern recognition - this method was planned to use microphones data histogram to recognize different commands. The shapes of recorded waves would be compared to decide whether they represent the same command. The commands would be detected as short periods of sound followed by and preceded by silence.

The method has significant advantages over the remaining two. Voice commands would be the most natural and result in better immersion as it would mimic interaction with real pet. Although speech recognition is not used, it would create a similar authentic experience for users. It would also allow creating more new commands than pose detection due to restricted number of distinct poses.

The major problem however would be data storage. Even with minimum duration of voice command a large amount of data would have to be stored and processed for each of them. For example assuming a command can be given in 3 seconds, with two of Miro's 20kHz microphones [1] a total of 120000 numbers would have to be stored for each command as well as currently detected sound. After several commands being learned this could massively affect performance of the software. Although reducing the number of collected samples could solve this issue, it might also reduce the accuracy of comparison.

Additionally the wave comparison would have to take time displacement between two commands into consideration. As the commands would likely not be detected at the exact same moment and their speed would not be the same, a simple pattern comparison would not be possible. Dynamic time warping - an algorithm used in speech recognition to measure similarity between two patterns [9] - could be used for this purpose.

The last disadvantage of this method would be sensitivity to noise. Applying a noise reduction algorithm would be necessary for the program to be used in regular household conditions.

To assess whether use of sound pattern recognition would be possible for the project, analysis of microphone data has been performed. Various commands have been recorded using robot's microphones and chart representation of the data has been created.

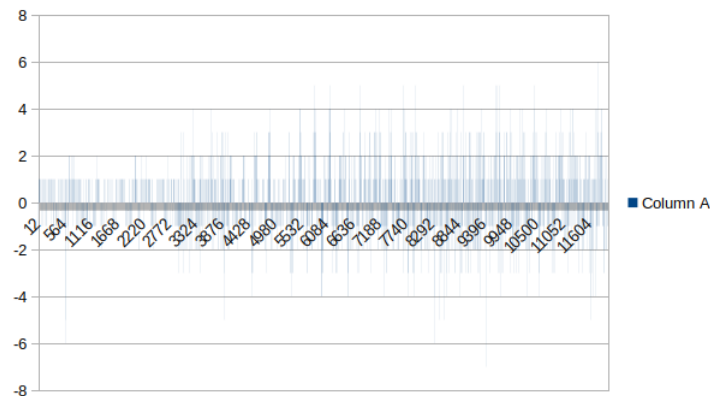


Figure 1.1: Example sound data

The chart on the Figure 1.1 presents raw unmodified data collected for one of the commands. For comparison purpose the values were converted into their absolute equivalents and the data has been reduced by dividing into equal parts and selecting maximum value from each of them.

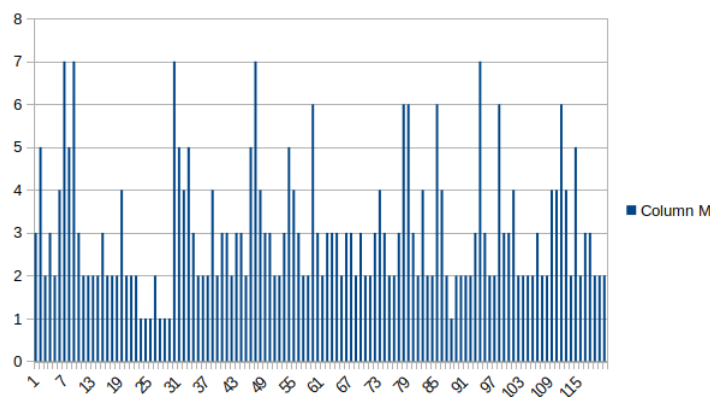


Figure 1.2: "Turn left" command- first recording

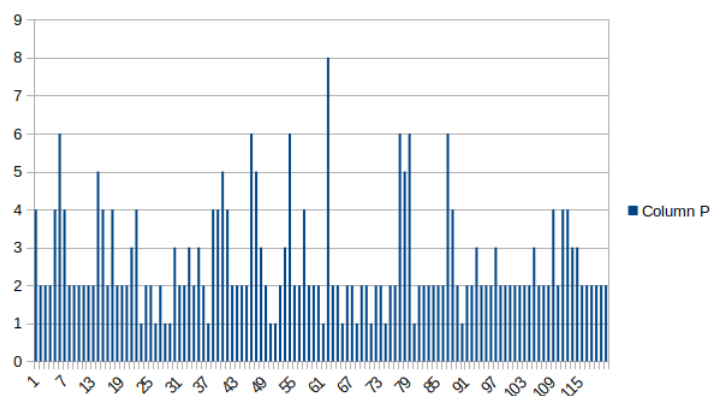


Figure 1.3: "Turn left" command- second recording

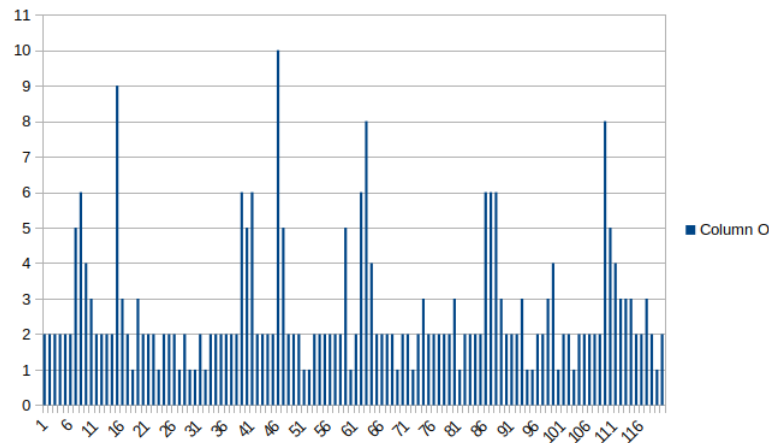


Figure 1.4: "Move forward" command

In the experiment two recordings of the same command have been compared to each other and a recording of different command. Expected result would be that the same command recordings patterns would be similar to each other and significantly different to the other command. However as can be observed on Figures 1.2, 1.3 and 1.4 this is not the case. To ensure the results were not caused by faulty recordings, the experiment has been repeated several times. As the results remained the same, voice pattern comparison was decided not to be a viable method of command recognition.

Pose detection - this would use the posenet library to detect and compare poses from the video data.

A considerable advantage of this choice would be the use of existing library. As most of the code would be already created and tested, implementation of command detection would only require slight modifications and integrating existing features into the system. Therefore more effort could be put into perfecting the learning system.

It would also allow easy representation of the commands in the monitoring system as the pose skeletons. Displaying a command in a visual form with description of corresponding action would simplify the control of the learning process.

The difficulty using this method would be selecting detection condition. Seeing a person from different angles would result in new commands being detected constantly without a condition for starting detection.

Moreover, the installation of the library would require administrator involvement and additional time for them to assess the necessity of that process. Therefore pose detection method has also been rejected.

Colour and shape detection - this method would use previously prepared cards with shapes and colours printed for them as commands.

Compared to the sound pattern recognition, implementation of this method would be much

simpler. Unlike the pose detection however, it would involve more contribution to the final system.

Recognition of commands on the captured video would also not be an issue. Comparison of commands could be initiated by detecting sufficiently large patch of one of specified colours that can be classified as any of available shapes.

Although the use of cards does not simulate interactions with animal well, it would be straightforward to learn and use for people of all ages.

The method provides the same advantage of visual representation as pose detection and possibly could be interpreted even more easily, as pose skeletons might be confusing especially for younger users.

The main problem acknowledged while choosing colour and shape representation of commands would be necessity of preparing physical copies of printed commands in advance.

Considering all above this method has been chosen for command detection.

1.2.4 Learning system

The choice of learning algorithm was clear immediately after the project direction of learning through interaction was decided. The idea of using positive feedback was inspired by emotion system existing in MiRo, but also the real life process of teaching pets. Reinforcement learning simulates this process better than any other available machine learning algorithm.

Another considered option was learning through imitation, however due to the robot's low quality cameras and significant movement limitations, this idea was shortly abandoned.

Firstly, the robot was planned to only be able to learn which action from an available set to choose as response to the command. The research about developmental learning for social robots [10] influenced the idea of building sequence of those actions as more complex tricks as well.

1.2.5 Other Choices

Programming language While working with ROS the possible choices of programming language included Python and C++. Although C++ offers better speed and multi-threading support which could be valuable for the project, it can also be difficult to learn and use properly. Due to large number of technologies not previously used being involved in the software, no time could be spared for studying the programming language. The simplicity and readability were a priority, therefore Python has been selected.

Real machine development vs simulation Two approaches were available for developing the software - using the real machine or the Gazebo simulation environment. Use of the physical robot could prove inconvenient in some aspects, as it required all work to be

performed in the Intelligent System Laboratory on campus, constant control of the battery level as well as additional caution not to damage the machine. However, as some of the robot's sensors were not included in the virtual environment, the decision has been made to develop the software on the physical machine.

1.2.6 Deliverables

- Documentation and instructions - containing user instructions for teaching MiRo and information about known issues
- MiRo learning software - files responsible for MiRo's behaviour and learning process
- Test scripts - a set of automated unit tests, and manual tests description with results achieved by the end of the project
- Final Report - discussion of created software, development process, techniques, third party libraries and tools used as well as scientific papers that have influenced work on the project

1.3 Process

Feature Driven Development has been chosen as the methodology to follow throughout the project due to natural identification of features during the research phase. The process has been adapted for a single person use where necessary. The work was started by creating a general design of the program and a simple class diagram. Previously identified set of features was then reviewed and improved where possible. A plan then was made on the order of implementing features and estimate time frame was assigned for each of them. As the project is an individual piece of work there was no need to assign feature owners as in regular FDD. Afterwards work on the features have started. Each feature involved a stage of further research and design, followed by implementation and testing.

The process has been organized and monitored using a Trello [11] board and GitHub repository has been used as a version control system for the project. The GitHub repository has been decided to be set as private as no suitable limited access options have been found - therefore no link is provided in the report.

Chapter 2

Design, Implementation and Testing

2.1 Overall Approach

2.1.1 Design

Following the process of Feature Driven Development, in the beginning an overall architecture has been developed. In order to create an initial class diagram, research has been conducted on Object Oriented design principles in the field of robotics.

Hardware abstraction was one of the main concepts advised to consider in the studied papers [12] [13], however due to the use of ROS providing a hardware abstraction layer, it would not be a major issue in the system. Furthermore, as the project is focused on developing software for specific robot - MiRo - its reusability on different platforms has not been considered to be a priority. Nonetheless the system was designed to prepare control messages in the same place, so that they can be easily modified and adjusted to another platforms needs.

The resulting class diagram can be observed on Figure 2.1

Following classes have been decided:

- Application - the main class of the program
- Robot - a class responsible for general robot's behaviour
- Command - a class in charge of representation and analysis of detected commands
- ActionManager - a class responsible for learning process and message preparation
- GUI - a class representing visual monitoring system

Creation of this diagram was focused on dividing the program into classes and deciding their main functions rather than delving into specific methods to be used within them. These were considered later in the project during the feature development process.

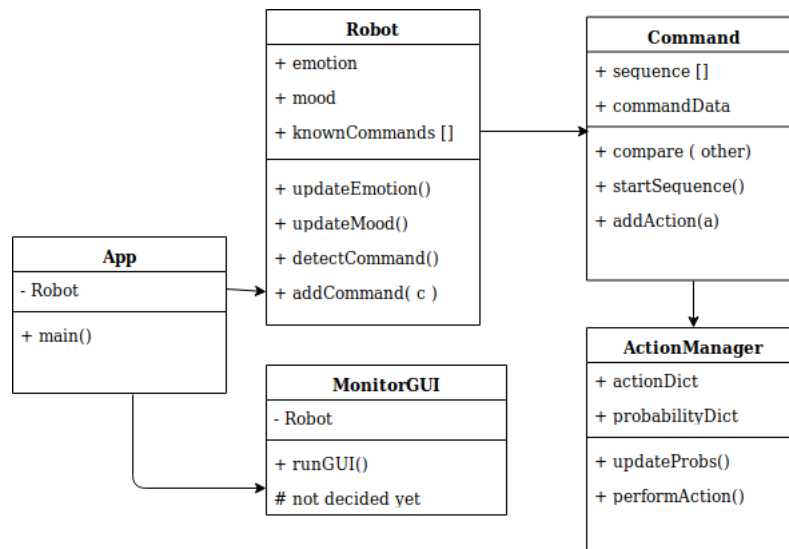


Figure 2.1: Initial class diagram

After the overall architecture has been developed, each feature included its own analysis of required functionalities and further design.

2.1.2 Implementation

Implementation of the MiRo's learning software has been done using jEdit text editor. Considering previous programming experience with more advanced IDEs offering multiple helpful functionalities such as error highlighting or typing autocomplete, work with a more simple editor was initially a challenging task. However with limited choice of Python IDEs available on university machines, jEdit offered the best readability of the code compared to other considered options.

As debugging in jEdit would require installation of additional plugins, command line debugger pdb has been used instead. Using a non-GUI debugger for the first time has also resulted in the process being considerably slower than expected.

While developing the program, several modules have been prepared - each responsible for one of the classes. Then, during feature implementation phase, code has been added to appropriate classes.

Whenever possible additional libraries and software have been used to simplify the software creation process.

2.1.3 Testing

The testing has been performed after implementation of each feature. Features have been tested first separately and then once more after integration into the system. Types of used tests included:

- Unit testing - where possible unit tests have been created for most of major methods using unittest and rostest test suite.
For testing methods including inner dependencies or time delays, use of mock testing framework has been considered. However due to older version of Python being used, no such framework was available in standard language libraries.
- Integration testing - parts of the system have also been tested together to ensure the correct interaction between the components
- Manual testing - for methods and behaviours impossible to test differently (such as those affecting actuator output), sets of manual tests have been performed
- User testing - during Aberystwyth Science Fair, the incomplete system has been presented to a number of young children, being one of the target age groups for the software. Although the system was missing multiple significant features at that time, the demonstration allowed for receiving a useful feedback from event participants. Additionally their interaction with the robot was observed, analyzed and used to adjust parts of the software design where necessary.

2.2 Actions

The first feature to be implemented was Actions - a feature responsible for simple behaviours used as parts of the tricks performed by the robot.

It was chosen for initial phase of the work due to its simplicity. It provided an opportunity to become familiar with the programming language, ROS and the robot while working on the implementation.

2.2.1 Design

Research on MiRo's actuators resulted in following actions being chosen to implement:

- Go - the robot moves forward for a certain distance
- Turn Left - the robot rotates anticlockwise
- Turn Right - the robot rotates clockwise
- Blink - the robot quickly closes its eyes and opens them again
- Lean Down - the robot leans its head towards the ground
- Rotate ears - the robot rotates its ears

Other considered options included barking, using LEDs on the robot back or closing and opening eyes based on their current state. The last option was planned to be used with previously considered voice command detection and would not be possible with visual detection system - therefore it was replaced with a quick blink.

The messages with appropriate information for robot's actuators would be prepared in ActionManager based on their probability. Initially probabilities for all actions would be equal.

To achieve this effect two dictionaries of actions were created - one responsible for calling appropriate methods that prepare the messages for platform control, and the other storing their probabilities. The probability dictionary is built based on the action dictionary to ensure the probabilities are always equal and their total value is 100 even if actions are added or removed in the future.

Classes and key methods:

ActionManager - a class responsible for storing probabilities of each action and performing appropriate action based on them

- performAction - method responsible for generating random action based on each action's probability. It generates a random number between 0 and 100 and decides which method to call based on this number. The pseudo code for the method is presented below.

Algorithm 1: PerformAction

```

1. Generate a random number n between 0 and 100;
2. foreach action a in probability dictionary do
    Calculate the probability sum of a and all previous actions;
    if n smaller than the sum then
        Prepare a message for a;
        break;
    end
    end
3. return message

```

- action methods - these methods include: go, headDown, turnLeft, turnRight, eyes, ears and are responsible for preparing the message for platform control to use in the Robot class

Robot - a class responsible for message handling

- performAction - sets the platform control message based on the result generated in ActionManager.
- callback_platform_sensors - a callback function that publishes previously prepared platform control message

2.2.2 Implementation

Initially a separate program has been written as a test of robot's capabilities. The program would take command line argument with a name of action to perform and the robot would perform the chosen action. After the results were satisfying the code was moved to the previously prepared classes.

With current implementation the robot is able to perform all actions specified above. Due to the chosen design however, it has not been possible to control duration of the actions separately. With the Robot class responsible for handling messages, the platform control message is being sent constantly as the program is running. Therefore the duration of action is only dependent on the time of resetting the message, regardless of which action it is. Because of that a specific distance to move by and a number of degrees to rotate by could not be chosen.

2.2.3 Testing

Due to the random nature of the main method implemented as a part of this feature, most of the testing performed was manual. The main aim of manual testing was ensuring the messages have been formed correctly to result in expected behaviour in the robot.

Some unit tests were also written by prior manual modification of probabilities for actions. These were performed to check that a correct action would be chosen based on the probability.

2.3 Emotion System

Another feature expected to be straightforward in design and implementation was the simple emotion system responsible for manipulating and representing MiRo's emotion and mood. The feature would then be used to influence the learning process.

2.3.1 Design

One of the main choices considered when designing the system was definition of emotion and mood in the program. Although in a realistic scenario emotions and mood would be affected by multiple factors and influence a range of different behaviours, such complexity was not needed for the purpose of the project. As the system was only meant to be used as a part of the learning software, a more simple solution has been designed.

The only factor to have an impact on state of the emotion was chosen to be user's interaction with the robot - more specifically touching it would result in positive emotions being invoked. A positive emotion would then be interpreted as a reward in the reinforcement learning algorithm.

The mood has then been defined as emotion over time. The purpose of the mood would be affecting the learning rate of the robot. As a result the robot would learn faster the longer the play session duration would be.

A need of visual representation of the robot's emotional state has also been identified when planning the elements to include in the feature. The movement of the tail has been chosen as the most natural and intuitive method of representing pet's happiness.

Both mood and emotion have been represented as single variables of Robot class. The emotion is constantly being updated according to values detected by touch sensors. If touch is detected on any of the sensors emotion is changed to positive, otherwise it is set to neutral. The mood is calculated every several seconds as the average emotion over time.

Another considered option was implementation of robot's preferences - some of the touch sensors would be randomly selected as favourites on the first run of the program. Touching them would then affect learning rate to a bigger degree than remaining sensors. Preferences would be stored in a text file and read when initializing the program. This was however planned as additional possible extension of the existing system and not its necessary element. Therefore, due to lack of time by the end of the project, it has not been included in the software.

Classes and key methods:

Robot - the emotion system has been designed as the Robot class only feature, this class is fully responsible for managing emotion and mood

- `updateEmotion` - sets the emotion to a correct value, based on the touch sensors output. Emotion is also added to the list of recent emotions.

- `updateMood` - increases or lowers the mood value based on the average emotion over time. The average is calculated using the list of recent emotions of a specified size.

2.3.2 Implementation

Although implementation of this feature has not entailed any major problems, the choice of emotion's representation has been changed several times. Initially both mood and emotions were planned to be represented by values between 0 and 1. The emotion value would be increased when the robot is touched and decrease otherwise. The same value would be used to set the tail movement rate.

During the Science Fair demonstration however, such representation appeared not to be intuitive especially for younger children. As the emotion growth would be dependent on the duration of touching the robot, the tail movement would only become observable after some time of keeping the hand on the touch sensor. Petting the robot like a real pet, would often not be able to increase the emotion value enough for the tail to start moving.

In order to improve the experience the emotion value has been decided to only take values of 1 - when the robot is touched and 0 otherwise. This would result in a quick and easy to notice reaction of the robot.

Another change from initial design worth mentioning was a method of calculating the mood. Initially the mood was planned to be calculated as the average emotion over time. However as the emotion is only dependent on the robot being touched, the positive emotions would be much more rare than negative (or rather neutral) ones. As a result it was difficult to significantly increase the value of the mood, and even after achieving it, it would quickly drop back to its original state. Therefore a new way of calculating the mood had to be prepared.

In the new implementation, although still dependent on the average emotion over time, the mood value was decided to increase fast and drop slowly. This is because the mood would be dropping for most of the time, and only be increased rarely during events of touching the robot. Such solution allowed for the mood to be more stable and balanced throughout the play session.

2.3.3 Testing

Unit test have been created for both of the updating methods. The tested cases verified whether the methods behave correctly in regular situations as well as on boundaries, e.g. trying to increase the mood when it already is at its maximum allowed value. Additionally a test has been performed checking whether the methods work correctly together.

The Science Fair has been used as an opportunity to perform some user testing by allowing the participants to interact with the system. This resulted in updating the representation of the emotion as described in the implementation section before.

2.4 Command Detection

Command detection was one of the most complex and challenging to implement features in the project. It is responsible for processing camera image, recognizing commands and responding to them.

2.4.1 Design

Commands have been decided to be represented by shapes and colour. For simplicity three possible shapes and three colours have been selected: square, triangle and circle in red, green or blue colour. Additionally commands detected with right camera would be interpreted as different commands than the ones detected with left camera. Such representation would allow for 18 separate commands to be taught while keeping the implementation relatively simple.

To increase the number of possible commands, additional colours and shapes could be introduced. However the more shapes and colours were added, the smaller the difference between them would get. Therefore recognition of commands would get less reliable.

Another option allowing more commands to be taught, could be using both cameras to form a single command. With 9 options to choose from for each camera it would result in a total of 81 possibilities. This approach however could result in timing issues while presenting the commands as both commands would have to be placed in front of cameras at the exactly same time.

Ultimately it has been decided that 18 distinct commands would be enough for the purpose of the project, as the users would likely not be able to remember many more combinations of shapes and colours to use them efficiently.

The robot would store a list of known commands. It would be constantly switching between 3 states: looking for commands, responding to the last detected command and waiting for feedback. Upon detecting a new command it would be added to the list and a random response would be generated. Detecting a previously seen command would result in searching the list for the method and responding based on the previous feedback for the command.

Initial design included visual representation of detecting command for the first time by tilting robot's head. This was planned to imitate the reaction often observed in real dogs when they do not understand the command. The idea was eventually abandoned as damaged parts of the robot prevented the sideways movement of the head.

Classes and key methods involved:

Robot - responsible for storing the list of the known commands, receiving and preparing camera image, and responding to command.

- detectLeft and detectRight - two callback methods responsible for receiving camera images

- `detectCommand` - method responsible for detecting and responding to commands. It extracts the command information from the camera image and generates response to that command, based on whether it has previously been seen or not. If the command is seen for the first time it is added to the list of known commands.
- `respondToCommand` - responsible for publishing platform control message, based on the action to be performed. The robot only responds to last detected command which is reset after action has been performed. This is done to avoid responding to a single command several times.

Command - responsible for storing responses to commands and analyzing command information on the images

- `getCommandData` - a method responsible for general shape and colour detection. It sets the command information stored in the object to correct values by delegating smaller tasks to appropriate detection methods. The overall algorithm for recognizing commands is described below.

Algorithm 2: Command detection

1. Preprocess the image by adjusting brightness and saturation;
 2. Apply a mask to discard the data of pixels with colours other than specified
 3. Find contours in the resulting image
 4. Identify the largest shape
 5. Classify the shape into one of specified shape categories
 6. Find the colour in the middle of the detected shape
 7. Classify the colour into one of specified colour categories
-

- `prepareImage` - method responsible for preparing an image for the command detection. Firstly the saturation and brightness of the image are adjusted for easier colour detection. Afterwards a mask is applied that removes all parts not falling into any of specified colour ranges. A result is a black and white image where white parts represent pixels that were either red, green or blue in original image and remaining pixels are set to black.
- `detectShape` - uses previously prepared image to find the largest shape and detect its contours. To avoid recognizing noise on the image as commands only contours with large enough area are further processed.
- `recognizeShape` - based on detected contours, classifies the shape into one of three possible options: a triangle, a square or a circle. The classification is performed using approximate number of vertices.
- `detectColour` - uses original image and checks the colour inside detected contours. The colour is then classified as red, green, or blue

- isCommand - ensures that all parts of the command were successfully identified. If any of them is missing the command is not valid and should not be treated as command.

2.4.2 Implementation

The commands have been represented by cards with shapes of different colours - the image presenting the cards used can be found in Appendix C.

As the command detection system was decided to involve some simple computer vision, OpenCV - an image processing library has been used to reduce amount of necessary work and simplify the implementation process. The library offers many useful tools for both colour and shape detection. Moreover it is well-documented and easy to use even without previous experience.

Current implementation of command detection feature allows for successful recognition of colours and shapes on images. Example output created by the system can be observed on Figure 2.2

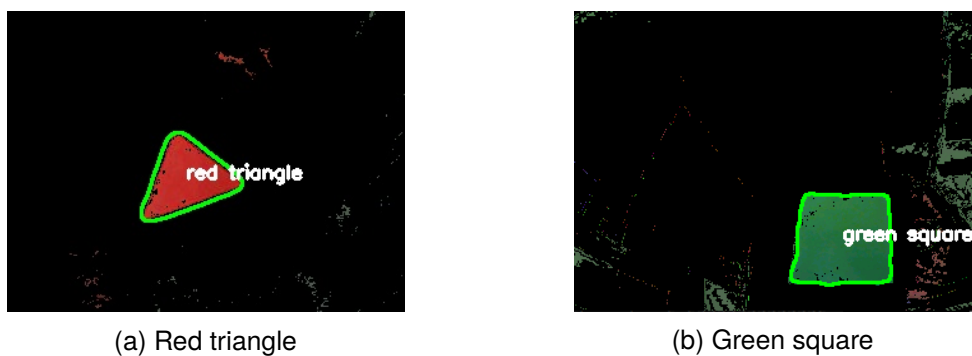


Figure 2.2: Output of the command detection

Comparison of the input image and resulting detection is presented on Figure 2.3

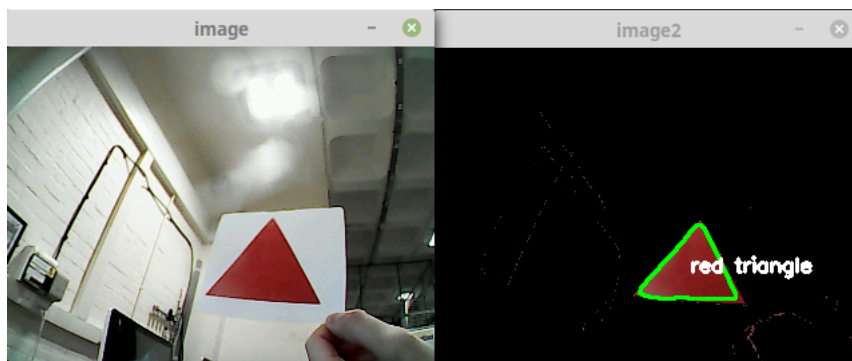


Figure 2.3: Comparison of input and output

The system is also capable of managing the list of known commands and responding to

detected commands according to the stored information.

A significant issue occurring during implementation phase was a repeated response generated for the same command. While responding to a command the camera image would not get updated and as a result after the action was finished, the same command would be detected again. The bug was eventually fixed by performing the response in a separate thread so that the camera image can still be updated and processed while executing the action.

Although the system developed for this feature implements all the necessary functionalities, it is by no means perfect and multiple issues with its implementation and functioning can be identified.

One of other main problem with using computer vision based system for a robot is the difficulty including initial colour calibration without providing additional software. However skipping this step might massively reduce the accuracy of command detection in varying lighting conditions. With the current design the ranges of acceptable RGB values are preset for specific colours - this sometimes results in parts of shapes not being detected and as a result the shape is not identified correctly. Although addition of the calibration software was not considered as a part of the project, the monitoring system could be extended by this feature in the future release.

Another issue with the current implementation is constant detection of commands. If the card with command is not moved to the camera quickly enough, the shape can be detected when only part of it is visible to robot resulting in incorrect classification. The example command detection output with the problem occurring can be seen on Figure 2.4. Although the presented shape is a triangle, a part of the figure on the right side of the image is not in the range of the camera. This causes additional edge to be detected and the shape is recognized as a square instead.



Figure 2.4: Detection with shape not fully in camera range

This problem could be fixed by introducing an additional condition for starting the command detection process. A separate sensor such as sonar located in MiRo's nose could be used for this purpose - an example solution would be detecting commands only when the robot's nose is touched.

2.4.3 Testing

Initially the system has been tested on previously prepared static images. The images represented different shapes and colours but represented perfect conditions rather than

expected working conditions with the shape being always placed in a correct position, so that full figure would be included in the picture.

This approach allowed for testing parts of the command detection, such as selecting appropriate colour boundaries or detecting the correct number of edges visible in the picture, however it did not take possible problems with camera image into account.

A better approach would be taking pictures randomly while moving the command card in front of the robot's camera, to simulate the intended way of using the commands better. Such tests would allow for identifying many of the remaining issues in the early development stage and as a result allocating additional time to find and implement an appropriate solution.

The results of detecting a new command compared to an already known command and choice of the correct response were tested manually as creating unit tests for these parts of the software would require the use of a mocking library.

Additionally some user tests have been performed during the Science Week. The simplicity of the process of presenting commands to the robot has been assessed based on participants of different age groups performing the task.

2.5 Learning System

This feature is responsible for the robot's learning process. It involves collecting feedback from users and using it to modify probabilities of performing each action after seeing a specific command.

2.5.1 Design

Although the learning technique used in the project is often referred to as reinforcement learning, this is mostly due to the positive reinforcement being used as a part of the teaching process - it does not include representation of states or actions like a typical reinforcement learning algorithm.

This is because the learning problem covered in the software is much more simple and does not require a complex state representation. The actions are learned separately and are treated as the outcome, rather than being chosen to lead to a specific state.

The learning process is conducted in the following way:

Initially seeing a command would result in generating a random response. After responding to the command the robot would switch into state of waiting for feedback for several seconds.

Providing a positive feedback by touching the robot would then affect emotion and increase probability of performing the same action in the future when detecting the same command. No positive emotion in the waiting period would be understood as negative feedback and reduce the chance of the action being chosen again.

The learning process for an action is finished when it reaches the probability of 100%.

Classes and key methods involved:

Robot - this class is mainly responsible for collecting feedback and initiating the process of updating probabilities

- `respondToCommand` - this method has been updated to wait for feedback after performing an action and updating the probabilities using it
- `collectFeedback` - this method causes the robot to check its emotion state for several seconds. If positive emotion is invoked at any point during that time, the feedback is set to positive.

ActionManager - this class is responsible for recalculating probabilities for the actions

- `updateProbs` - using provided learning rate updates the probability of the last performed action. Then it recalculates probabilities of remaining actions so that the total value of all probabilities is always equal to 100. The probabilities are affected proportionally to their previous share in the total probability. The probability is not updated if it is already equal to 100

Algorithm 3: Update probabilities

1. Calculate the sum of learning rate (LR) and probability of chosen action (p_A);
 if *sum larger than 100* **then**
 | Recalculate LR as $100 - p_A$;
 end
 2. Calculate rest (R) as sum of all probabilities except for p_A ;
 3. Update p_A as $p_A + LR$
 4. Calculate new rest (nR) like previously
 5. Calculate rest ratio (RR) as nR / R
 6. **foreach** *action B* **do**
 if *B is not chosen action* **then**
 | Update probability of B (p_B) as $p_B * RR$
 end
 end
-

2.5.2 Implementation

Although significant amount of time was spent finding the right mathematical formula for updating probabilities correctly, regardless of learning rate being positive or negative, otherwise implementation of this part was less challenging than expected.

A change to the original design has been made when implementing the feedback collection. Originally the probabilities were planned to be updated constantly as the feedback is collected. However this would result in the rate of learning process being affected by the duration of touching the robot. To avoid that, the process has been changed to only detect whether the positive feedback has been provided during the feedback collection. The probabilities would then be only updated once after the feedback collection is finished.

One of the issues that could be improved is the behaviour of the robot when the learning process is finished. Although the probabilities are not updated anymore after they reach the value of 100, the feedback is still being collected for them. This results in an additional waiting time while performing the trick, which could be skipped if an appropriate condition was added.

An important aspect to be noted about emotion and feedback is that, although they are both affected by touch eventually and may seem as the same thing, they cannot be used interchangeably. The feedback is only collected after action has been performed based on the emotion. Emotion on the other hand is updated constantly. Therefore touching the robot while it is looking for commands can be used to improve its mood and effectively the learning rate before starting the teaching process.

2.5.3 Testing

The `updateProbs` method has been tested thoroughly with multiple unit test cases. The tests covered a regular working behaviour as well as possible error conditions. Probabilities of both selected action and remaining actions have been verified after the use of the method.

The tests allowed to validate correctness of the resulting behaviour when updating probabilities with both positive and negative learning rate. They also verified whether the probabilities were updated correctly on the edge values, for example ensuring the probability would not be increased to over a 100.

Other tests included manual checking whether the correct action is being updated.

2.6 Sequence Building

This feature allows for creating more complex tricks formed of sequences of simple actions.

2.6.1 Design

For this feature existing code design had to be adapted for performing not one but a sequence of actions as a response to the detected command. Therefore the Command class was modified to store a list of responses rather than a single ActionManager object.

Introducing a sequence building system required additional conditions for extending and finishing the sequence. The chosen method of managing this process was creating a new type of action - a "stop action".

Although the "stop action" is created as regular ActionManager instance, it is not capable of performing any actions other than "stop" which does not affect the robot's behaviour except for resetting the platform control message. By default ActionManager objects are created as regular actions. The "stop" action is created by passing additional boolean parameter to the ActionManager constructor. A better design could include use of inheritance - regular and stop actions extending a general action class.

The "stop action" allows for collecting feedback about ending the sequence. Initially a response to any command consists of one random action and one "stop action". The feedback provided after "stop action" allows to control the length of the sequence. Positive feedback keeps the sequence length the same, while no feedback allows for extending it by another action.

Classes and key methods involved:

Robot - responsible for performing the sequence of actions as well as controlling the sequence length

- `respondToCommand` - this method was modified to perform all the stored action responses followed by a period of waiting for feedback. If the feedback received for the "stop action" was negative, it would then replace it with a regular random action and add a new "stop action" at the end of the sequence.

ActionManager - used for creation of "stop actions" as well as performing them

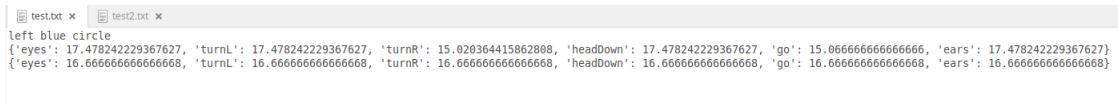
- `stop` - resets the platform control message to its initial state

The sequence being performed in the Robot class rather than Command is not a perfect solution as the Command's instance variables are accessed from outside the class. Separating the code into methods responsible for parts of the sequence building process and delegating the work to be done inside the Command class would be a better option, however as the feedback has to be collected after every action within the Robot class, no appropriate solution has been found.

2.6.2 Implementation

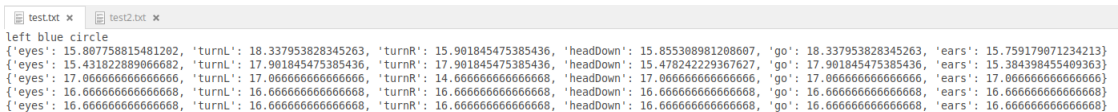
The implementation of this feature was relatively simple as the only functionality requiring an entirely new solution was extending the sequence by a new action. This has been easily achieved by using the feedback to the “stop action”.

The effect can be observed on Figures 2.5 and 2.6. In both cases the robot has been shown a command several times. In the first case it has been always rewarded after the “stop action”, in the second - no reward has been given after the “stop action”.



```
test.txt x test2.txt x
left blue circle
{'eyes': 17.478242229367627, 'turnL': 17.478242229367627, 'turnR': 15.020364415862888, 'headDown': 17.478242229367627, 'go': 15.066666666666666, 'ears': 17.478242229367627}
{'eyes': 16.666666666666668, 'turnL': 16.666666666666668, 'turnR': 16.666666666666668, 'headDown': 16.666666666666668, 'go': 16.666666666666668, 'ears': 16.666666666666668}
```

Figure 2.5: Positive feedback to “stop action”



```
test.txt x test2.txt x
left blue circle
{'eyes': 15.807758815481202, 'turnL': 18.337953828345263, 'turnR': 15.901845475385436, 'headDown': 15.855308981208607, 'go': 18.337953828345263, 'ears': 15.759179071234213}
{'eyes': 15.431822889066668, 'turnL': 17.901845475385436, 'turnR': 17.901845475385436, 'headDown': 15.478242229367627, 'go': 17.901845475385436, 'ears': 15.384398455409363}
{'eyes': 17.066666666666666, 'turnL': 17.066666666666666, 'turnR': 14.666666666666668, 'headDown': 17.066666666666666, 'go': 17.066666666666666, 'ears': 17.066666666666666}
{'eyes': 16.666666666666668, 'turnL': 16.666666666666668, 'turnR': 16.666666666666668, 'headDown': 16.666666666666668, 'go': 16.666666666666668, 'ears': 16.666666666666668}
{'eyes': 16.666666666666668, 'turnL': 16.666666666666668, 'turnR': 16.666666666666668, 'headDown': 16.666666666666668, 'go': 16.666666666666668, 'ears': 16.666666666666668}
```

Figure 2.6: Negative feedback to “stop action”

As presented on Figure 2.5 positive feedback allows to keep a constant length of the sequence, while negative feedback presented on Figure 2.6 extends the sequence by another action. The last action on both images is a “stop action”.

Apart from that the already existing parts of the code had to be modified to manage a list rather than a single action, which - in most cases - was not a big issue.

The resulting system is capable of controlling the sequence length, updating probabilities for actions in the sequence correctly and performing a sequence in which each of the actions is followed by a period of waiting for feedback.

2.6.3 Testing

The testing has been performed by verifying the length of the sequence based on the feedback provided.

Additional manual tests have been performed to validate whether all actions in the sequence are being performed.

2.7 Monitoring GUI

This feature was designed for easier control of the learning process. It would provide a visual representation of the known commands and the system's state. for controlling learning process

2.7.1 Design

The GUI would be responsible for displaying all the information useful for monitoring the learning process. It would provide an easy access to the emotion, mood and current state of the robot. The states would represent the current action of the robot such as looking for commands, responding to command or waiting for feedback. While responding to command an information would also be given on which command has been detected and which action has been chosen as a response.

Additionally a list of known commands would be displayed. Commands would be represented by images of detected shapes. For this purpose every time a new command is detected an image would be saved. The list would also include a button with a text description for each command. Clicking the button would allow user to display sequence of actions associated with chosen command, and probabilities for each of the actions.

Initial design of the layout of GUI is presented on Figure 2.7

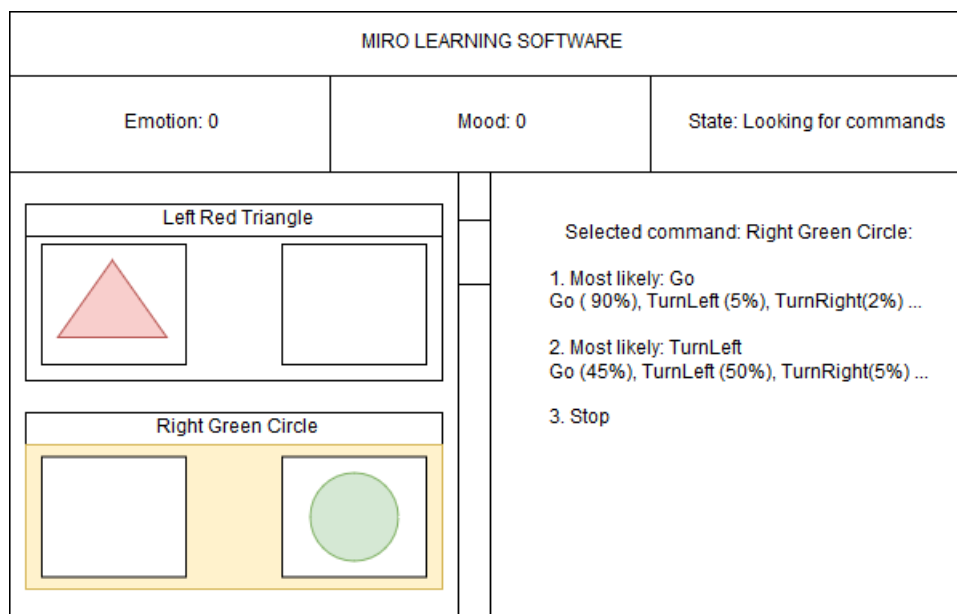


Figure 2.7: Initial Design

After further consideration the current camera image and some simple instructions on how to use the software have also been added to the GUI. The camera real-time camera image would be replaced with robot's interpretation of the command when a command is detected.

Classes and key methods involved:

GUI - the class responsible for preparation of graphical interface and keeping it updated

- `update` - this method refreshes the interface by updating the robot's state information as well as appending newly detected commands to the visual representation of known commands list
- `on_button_clicked` - displays information about actions associated with the command based on the text label set on the button creation

Some minor changes to other classes have also been made. These included e.g. saving images on first detection of specific command or preparing a text description of current state of the robot.

2.7.2 Implementation

There were multiple libraries available for GUI creation and interaction. Choice of GTK was mainly caused by the fact the MiRo Developer Kit included an GUI example using this specific library. Therefore it was considerably easier to learn and identify potential problems, using the provided example program as a guide.

Another solution inspired by the MDK GUI program was the use of Glade tool for creating an overall structure of the interface. Designing the structure this way allowed to simplify the process and significantly reduce the amount of time spent managing the layout of included containers.

Compared to original design the implemented GUI contains all the same elements and additionally offers the real-time camera view. Another difference between them is lack of clickable area behind the command representation as shown of Figure 2.8. The information about the command is displayed by clicking a button next to the command instead.

One of the significant issues with the implementation of the GUI is the way the currently seen camera image is displayed. As this part of the interface was added at the very end of the project with little time left for debugging and testing it was implemented in an extremely inefficient way - the camera image is saved into a file and reloaded every time the GUI is updated. This is because the GTK layout containers can store neither OpenCV image type nor ROS image type used in most parts of the software. They can only load an image from the file or a Pixbuf. An attempt has been made to convert the OpenCV image into a Pixbuf and use it to display current camera view - this was however unsuccessful. Although the solution does not seem to slow the system down significantly, it would be worth reworking it if additional time was available.

2.7.3 Testing

Due to simplicity of the graphical interface and lack of time remaining by the end of the project, the GUI has only been tested manually by verifying that all information is displayed correctly.

Images of the GUI working are presented below.

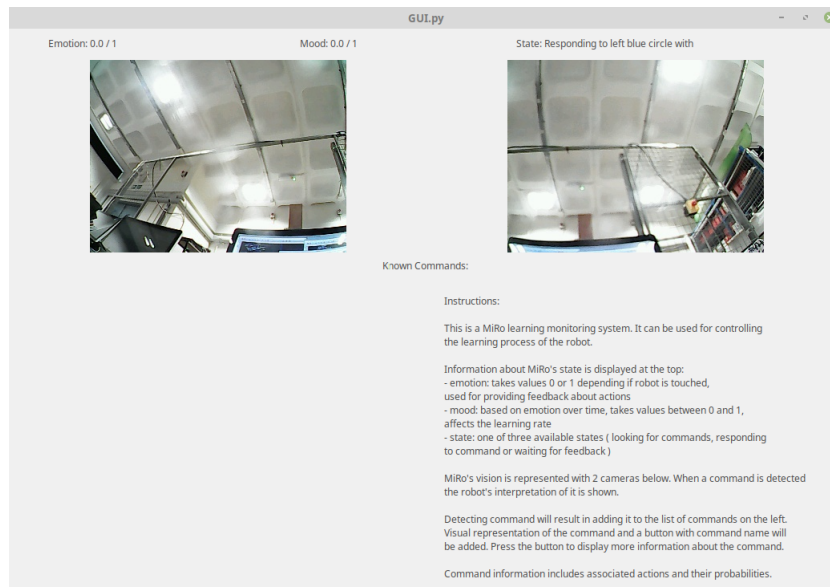


Figure 2.8: GUI: Initial State

Figure 2.8 shows the GUI in its initial state with no commands detected yet and an instruction of how to use the monitoring system displayed.

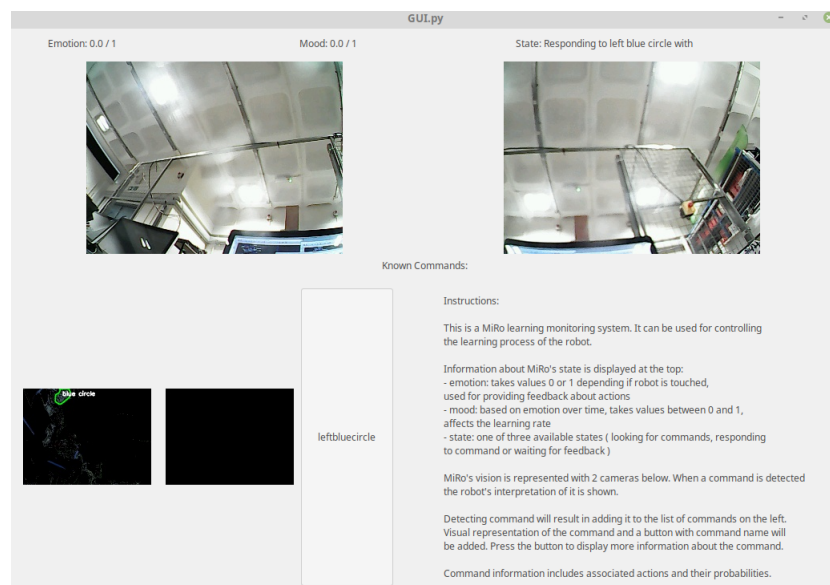


Figure 2.9: GUI: New command added

Figure 2.9 presents the interface after a new command has been detected. Its visual representation is added to the list of commands on the bottom left side together with a button with the command's name

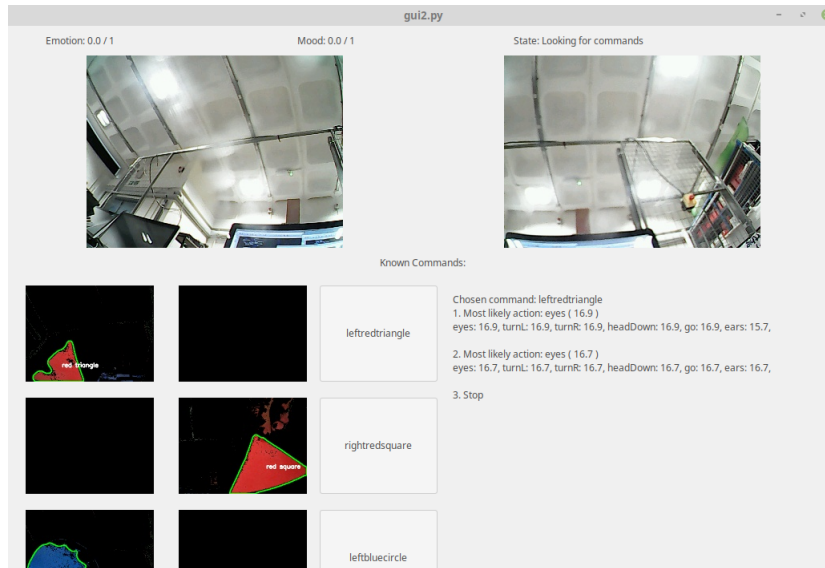


Figure 2.10: GUI: Command Information

Figure 2.10 depicts the monitoring system after pressing the button next to the chosen command. A list of associated actions and their probabilities is displayed instead of the initial instructions.

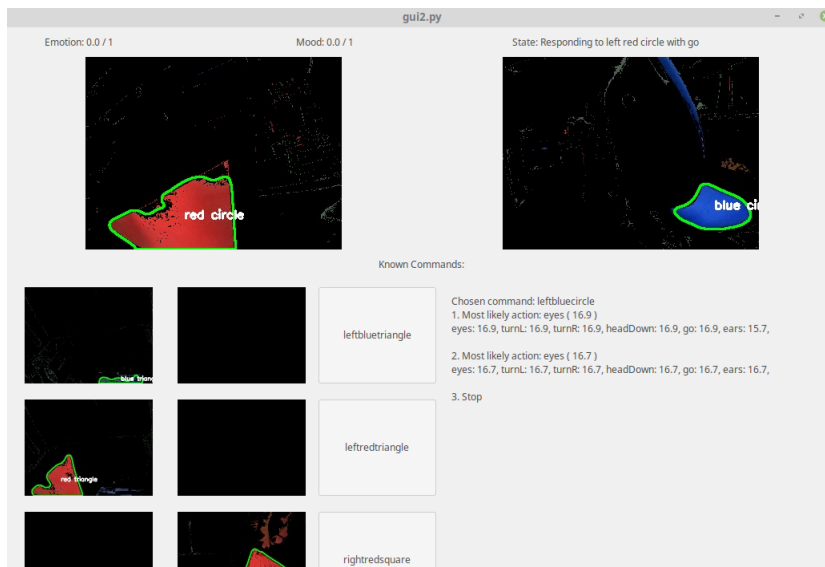


Figure 2.11: GUI: Command currently detected

Figure 2.11 presents the GUI state when the command is presented to a robot. Instead of the regular camera image, the robot's interpretation of the command is displayed in the current camera image section. In this case two commands are being shown at the same time - one to each of the robot's cameras.

2.8 Final Design

The final design of the system is presented on Figure 2.12.

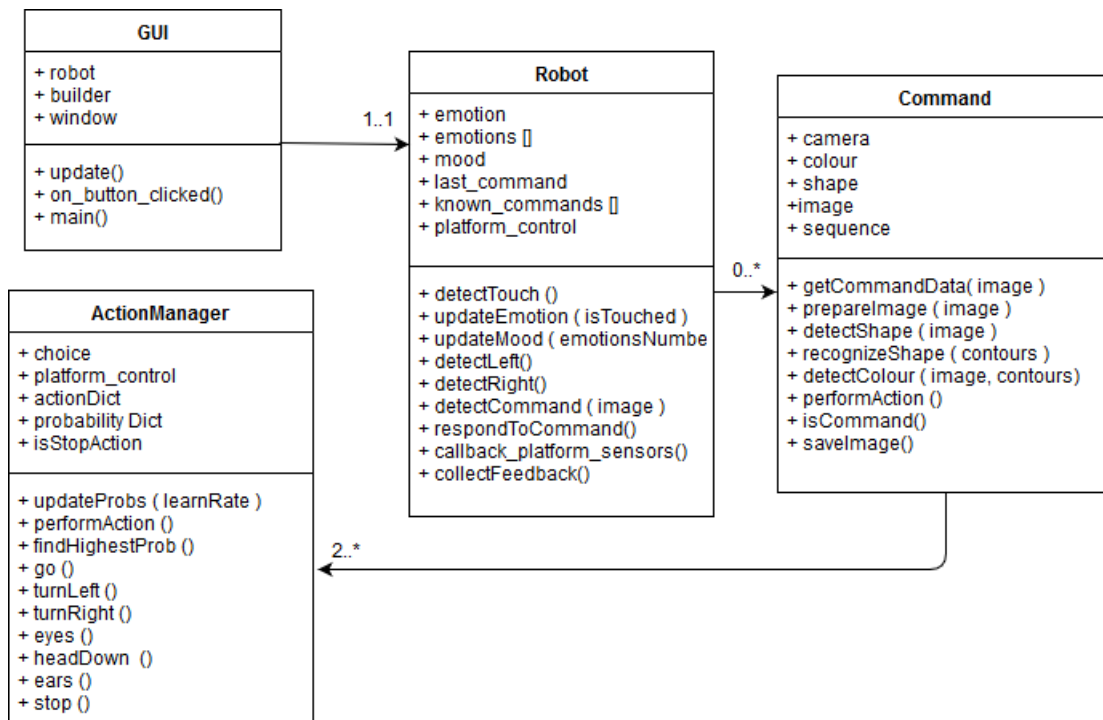


Figure 2.12: Final system design

As can be observed the Application class from the initial design has been removed and the class responsible for the main method is GUI. The application class was decided not to be necessary as it was only used for the purpose of running the main method.

Apart from that no changes to the general structure were made compared to the original design. As the features were developed the classes were filled with appropriate methods and instance variables.

Chapter 3

Evaluation

3.1 Analysis and research

The research conducted before starting the work has resulted in defining a project with a broad potential. A learning through interaction system for a robot could easily find an application in the robotic toys and companions industry. It has been a choice relevant for the studied Computer Science and Artificial Intelligence scheme, however it has also included significant elements of other related fields, such as robotics, not studied before. Effectively it has allowed for the project to not only be an opportunity to apply the previous knowledge in practise, but also gain experience in different fields.

The choice of tools used in the project has generally been satisfactory. Python has been an excellent programming language to use for the system development, as it was relatively easy to learn which allowed for more time to be spend on other aspects of the project. The choice of IDE, however, could definitely have been better as lack of expected tools slowed down the process considerably. Possibly spending some additional time researching available options in the initial stage, would result in much more time being saved during the development.

Feature Driven Development worked well as a methodology to follow in the project, mainly because it provides more freedom when setting the length of the iterations compared to other agile approaches. As some parts of the system required significant amount of time and others could easily be developed in a much shorter period, the flexibility of FDD has been extremely useful. The methodology has been followed consistently throughout the project and the features to be implemented have been selected well considering the time available for development.

The time management during the project was not perfect - although the initial plan divided available time considerably well between planned features, more time should have been arranged for the learning process especially in the initial stage. With a large amount of new technologies and tools used in the project, development has always seemed to be slightly delayed compared to the original plan.

3.2 Design, Implementation and Testing

3.2.1 Design

The design of the project has generally been solid. The choice of the object oriented design allowed for dividing the system into smaller, more manageable parts.

The requirements for each feature have been identified correctly - they have been clear, complete and contained a good level of detail allowing to avoid confusion when implementing specific functionalities.

An attempt has been made to create an easily maintainable design and keep the system open for extensions. As an example, removing or introducing additional actions into the system would not be an issue, due to the adaptive design of probability related parts.

Many decisions have been corrected or improved compared to the original ideas resulting in a better user experience. However there has also been multiple design choices made, that could be significantly improved, as they are causing a relatively high coupling between some of the modules. An aspect that should have definitely been improved is accessing and modifying elements of other classes present in the sequence building related methods.

Additionally considering the degree scheme studied a more complex learning algorithm could have been chosen, or a machine learning technique could have been used to improve the computer vision involved in command detection.

3.2.2 Implementation

Considering the time frame of the project and the amount of new technologies used in its development, the necessary functionalities for each feature have been implemented reasonably well. None of the requirements are missing, and although some of the features are not working as well as expected, they are all usable and form a final system extremely close to the one aimed at.

The good choice of additional libraries and software used in the project has allowed for the features to be developed more easily, as they provided sets of extremely useful tools reducing the time spent on implementing multiple parts of the system.

The effort has been made to keep the implemented code well organized and written in a consistent style. All major methods have been documented and described with comments to allow quick identification of their purpose. Whenever possible the methods have been divided to keep them responsible for a single task each.

Some of the aspects of implementation that could be improved on is focusing more on the efficiency of the program, as well as making more effort to reformat the code when needed.

3.2.3 Testing

The overall approach used for testing the system has been correct, however the process itself could be significantly improved.

The tests were written and performed consistently during development of each feature. They included multiple types of tests such as unit tests, integration tests, manual tests and in some cases user tests.

However not all of those types were performed for each feature. Often the main method of testing used was manual testing as creating unit tests without the use of a mocking framework has not been possible. With no such framework available, more effort could have been focused on the integration testing instead. Additionally the results of the manual testing could have been documented better, including photos or screenshots of the obtained results.

Although some parts of the program, such as the learning code, have been tested properly considering regular working conditions as well as error conditions, generally the system has not been tested comprehensively.

3.3 Future Work

Even after the project is finished many parts of the system would still need improvement to be effectively used in the industry.

One of the main issues of the current software is command detection accuracy. This should be addressed as it can cause the teaching process to be a tedious task rather than a fun interaction it was intended to be. The possible solutions to the problem have been described before in the Command Detection section of the report.

Another part that could have been made better is the GUI. In its current state it does not seem to be completely intuitive so redesigning it to provide better access to important information would be a good idea. Additionally efficiency of the real-time camera image could be improved.

After resolving the issues with the current system it could be used as a part of a larger, general smart robot companion software or adapted and applied to different types of robots.

Annotated Bibliography

- [1] Consequential Robotics, “MiRo Platform Interface,” <https://consequential.bitbucket.io/TechnicalInterfacesPlatformInterface.html>, 2019.

Information about MiRo’s sensors and actuators

- [2] —, “MiRo Core Interface,” <https://consequential.bitbucket.io/TechnicalInterfacesCoreInterface.html>, 2019.

MiRo’s documentation with information about the existing emotion system

- [3] “ROS Tutorials,” <http://wiki.ros.org/ROS/Tutorials>, 2019.

Set of ROS tutorials explaining basic concepts of topics, messages and nodes

- [4] K. Dautenhahn, “Robots we like to live with?! - a developmental perspective on a personalized, life-long robot companion,” in *RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No.04TH8759)*, Sep. 2004, pp. 17–22.

A paper describing socializing and personalizing companion robots

- [5] Sony, “AIBO,” <https://us.aibo.com/feature/feature1.html>, 2019.

Information about companion robot AIBO

- [6] —, “Teaching AIBO tricks,” <https://helpguide.sony.net/aibo/ers1000/v1/en-us/contents/TP0001970107.html>, 2019.

Instructions on teaching AIBO new tricks

- [7] SolveLight, “DST ROBOT – GENIBO SD COMPANION ROBOT DOG,” <https://www.solveilight.com/product/dst-robot-genibo-robot-dog/>.

Information about companion robot Genibo.

- [8] WowWee Robotics, “RoboPet user’s manual,” <http://www.theoldrobots.com/images7b/Robopet.Manual.pdf>.

Information about companion robot RoboPet.

- [9] Steve Cassidy, Department of Computing, Macquarie University, "Dynamic Time Warping," <http://web.science.mq.edu.au/~cassidy/comp449/html/ch11s02.html>, 2002.

Information about dynamic time warping and how it can be used for comparing patterns

- [10] A. Galdeano, A. Gonnot, C. Cottet, S. Hassas, M. Lefort, and A. Cordier, "Developmental learning for social robots in real-world interactions," in *HRI 2018*, 2018.

A paper describing developmental learning for social robots

- [11] Aleksandra Madej, "MiRo Trello board," <https://trello.com/b/qlv629KO/miro>, 2019.

Trello board used to control development of features for the project

- [12] M. Rickert and A. Gaschler, "Robotics library: An object-oriented approach to robot applications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 733–740.

A paper describing object-oriented approach design principles in robot applications

- [13] M. Löffler, D. Dawson, E. Zergeroglu, and N. Costescu, "Object-oriented techniques in robot manipulator control software development," vol. 6, 02 2001, pp. 4520 – 4525 vol.6.

A paper describing how object-oriented approach can be used for an example robotics system (Robot Manipulator Control Software)

- [14] Adrian Rosebrock, "OpenCV shape detection tutorial," <https://www.pyimagesearch.com/2016/02/08/opencv-shape-detection/>, 2016.

The tutorial used for implementing shape recognition for the command detection system.

- [15] JannisBushs, "PoseNet for ROS," https://github.com/JannisBush/ros_posenet, 2018.

A library for estimating human poses

Appendices

Appendix A

Third-Party Code and Libraries

Libraries:

- RosPY - The project used the ROS framework to communicate with the robot. The library is open source and is available from the ROS GitHub repository together with a set of other introspection tools. The library has been used without modification.
- OpenCV - The project involved simple computer vision elements for the command detection system. OpenCV provided useful image processing tools used in the implementation of that feature. The library is open source and is available from OpenCV website. The library has been used without modification.
- GTK - This library provided graphical interface creation and management tools used in the project for the GUI monitoring system. The library is open source and is available from The GTK Project website. The library has been used without modification.

Other third-party code:

- fmt function - This function is used in the project for formatting the accessed touch sensor data information. It has been copied from the `miro_ros_client_gui.py` program provided with the MiRo Developer Kit.
- shape detection - some elements of code from the Pyimagesearch shape detection tutorial [14] have been used and adapted for the needs of the project in the command detection system. The tutorial has mostly been used as a guide and no fragments of code of significant length have been directly copied into the software.

Appendix B

Ethics Submission

21/03/2019

For your information, please find below a copy of your recently completed online ethics assessment

Next steps

Please refer to the email accompanying this attachment for details on the correct ethical approval route for this project. You should also review the content below for any ethical issues which have been flagged for your attention

Staff research - if you have completed this assessment for a grant application, you are not required to obtain approval until you have received confirmation that the grant has been awarded.

Please remember that collection must not commence until approval has been confirmed.

In case of any further queries, please visit www.aber.ac.uk/ethics or contact ethics@aber.ac.uk quoting reference number **12408**.

Assesment Details

AU Status

Undergraduate or PG Taught

Your aber.ac.uk email address

alm82@aber.ac.uk

Full Name

Aleksandra Madej

Please enter the name of the person responsible for reviewing your assessment.

Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment

rrz@aber.ac.uk

Supervisor or Institute Director of Research Department

cs

Module code (Only enter if you have been asked to do so)

CS39440

Proposed Study Title

MMP Miro - a learning pet robot

Proposed Start Date

28/01/2019

Proposed Completion Date

03/05/2019

Are you conducting a quantitative or qualitative research project?

Mixed Methods

Does your research require external ethical approval under the Health Research Authority?

No

Does your research involve animals?

No

Are you completing this form for your own research?

Yes

Does your research involve human participants?

No

Institute

IMPACS

Please provide a brief summary of your project (150 word max)

This project will develop a trick learning system for the pet robot MiRo. The robot will be able to recognize new commands consisting of shapes and colours, and learn how to respond to them based on user's reaction. It will first respond to a new command with a random action and based on provided feedback (stroking or ignoring the robot) it will adjust the probability of performing the same action for this command. It will also be able to build more complex tricks by creating a sequence of simple actions. The project will also involve developing a simple GUI monitoring system to control robot's learning progress.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?

Yes

Does the research pose more than minimal and predictable risk to the researcher?

No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?

No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.

Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.

Yes

Please include any further relevant information for this section here:

Appendix C

Shape cards

