SOFTWARE ARCHITECTURE DESCRIPTION (SAD)

YOLO Robot

Gadir Suleymanli: 2416907
Muhammad Ebad Malik: 2414860

TABLE OF CONTENTS

List of Figures

List of Tables

1. Introduction

This document is the Software Architecture Description (SAD) document for the social robot YOLO (Your Own Living Object). The document includes architectural views of the robot, including context, functional, information and deployment views of the system.

1.1. Purpose and Objectives of the YOLO Social Robot

YOLO social robot is an educational robot designed specifically for children. The main purpose of the robot is to "interact" with the children and enable them to create imaginary stories by using their creative thinking skills. The children play with the robot for a certain period of time and the robot repeats their actions for a while. Afterwards, the robot carries out an unexpected movement based on child's previous actions, allowing the child to adjust and continue the story as they wish. The objectives of the system include: boosting the kids' creative abilities, letting them think "outside of the box", improving their social skills by engaging them in an interaction.

1.2. Scope

The scope of the system includes the following:

- The robot should be initialized and a certain social behavior should be set based on the kid's personality in order for the robot to start working. It is recommended to be done by an adult supervisor or parent;
- The robot repeats the child's actions until a certain period of time, allowing the kid to build up the story;
- The touch sensors in the system allow the robot to determine whether the child is touching it or not. If the child is touching and playing with the robot, then the movement information is stored within the system to be used later. Otherwise, the robot mirrors the child's movements based on the data collected;
- Later on, YOLO does an unexpected movement based on the previously recollected action history;
- YOLO requires connection to the router and the computer software in order to function properly. There could be some distance between the robot and router in order to allow the children to play outside of the house;
- The robot includes some LEDs connected to it. The color and brightness of the LEDs are set depending on the social behavior and the state of the touch sensors to allow better interactivity to the child;

1.3. Stakeholders and Their Concerns

The stakeholders that are involved in using and developing the system are as follows:

a. Children (main users): YOLO is ultimately made with and for children as they are the essential and primary users. Therefore, they play a pivotal role in the development of YOLO and are active contributors especially in the areas such as design, research and testing. The main concern of the children is the robot functioning properly and robots' actions fitting the story line, since it helps them build their stories.

b. Supervisors (Educators/Parents): They are a major stakeholder in the project as they are the guardians of the primary users; children. Therefore, their primary concern is the children's safety. As long as the child is safe and is not physically or emotionally damaged by the robot, supervisors' needs will be met. Moreover, cost of purchasing a YOLO is another concern for them, since they are the ones financing the robot.

c. Researchers: Their main purpose is observing children's actions and their interactions with the robot in order to be used in social and psychological research. They are mainly concerned with using the robot as a controllable and programmable instrument by the social and cognitive sciences sector to research the developmental characteristics of children while interacting with robots.

d. Developers: Apart from the main developers of the robot, the API allows the developers to add new behaviors to the system. Therefore, both development team and the external developers are involved with the system. They are mainly concerned with writing code that is secure and meets the users' requirements. They also want to make sure that the new features make the robot more user-friendly and more enjoyable and interactive for the children.

2. References

This document is written with respect to IEEE 42010 standard, using the source below:

[1] 42010-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering– Architecture Description.

Other Sources:

[2] Alves-Oliveira Patrícia, Arriaga Patrícia, Paiva, A., & Hoffman, G. (March, 2021). Children as Robot Designers. *HRI*, 399–408.

[3] Alves-Oliveira Patrícia, Arriaga Patrícia, Paiva, A., & Hoffman, G. (July, 2019). Guide to build YOLO: a creativity-stimulating robot for children. *HardwareX*.

[4] Alves-Oliveira Patrícia, Arriaga Patrícia, Gomes, S., Chandak A., Paiva A., & Hoffman, G. (March, 2020). Software Architecture for YOLO: a creativity-stimulating robot. *SoftwareX*.

3. Glossary

API	Application Programming Interface
KNN	K nearest neighbor is a Machine Learning Algorithm, which determines the likelihood of a point being a member of a particular group according to the group that are nearest to it are within.
LED	Light-Emitting Diode (refers to the lights on the robot)
ML	Machine Learning
Open Source Software	Type of software with source code that external people can view, edit and enhance
OS	Operating System
Raspberry Pi	A low-cost, small computer that can be plugged into a computer monitor, which enables people of all ages to learn about computing and how to program in programming languages, including Python and Scratch
Router	A networking device that forwards data packets between computer networks
YOLO	"Your Own Living Object" (name of the robot)

4. Architectural Views

4.1. Context View

4.1.1. Stakeholders' use of this view

In this particular viewpoint, context of the system is provided, with the actors and other systems related to it in a general manner with detailed explanations. Stakeholders, especially the users of YOLO robot, may make use of this view in order to understand different interactions that may occur between the robot and stakeholders, as well as impact of the system on its environment and how external entities and services are used from a context view.

4.1.2. Context Diagram

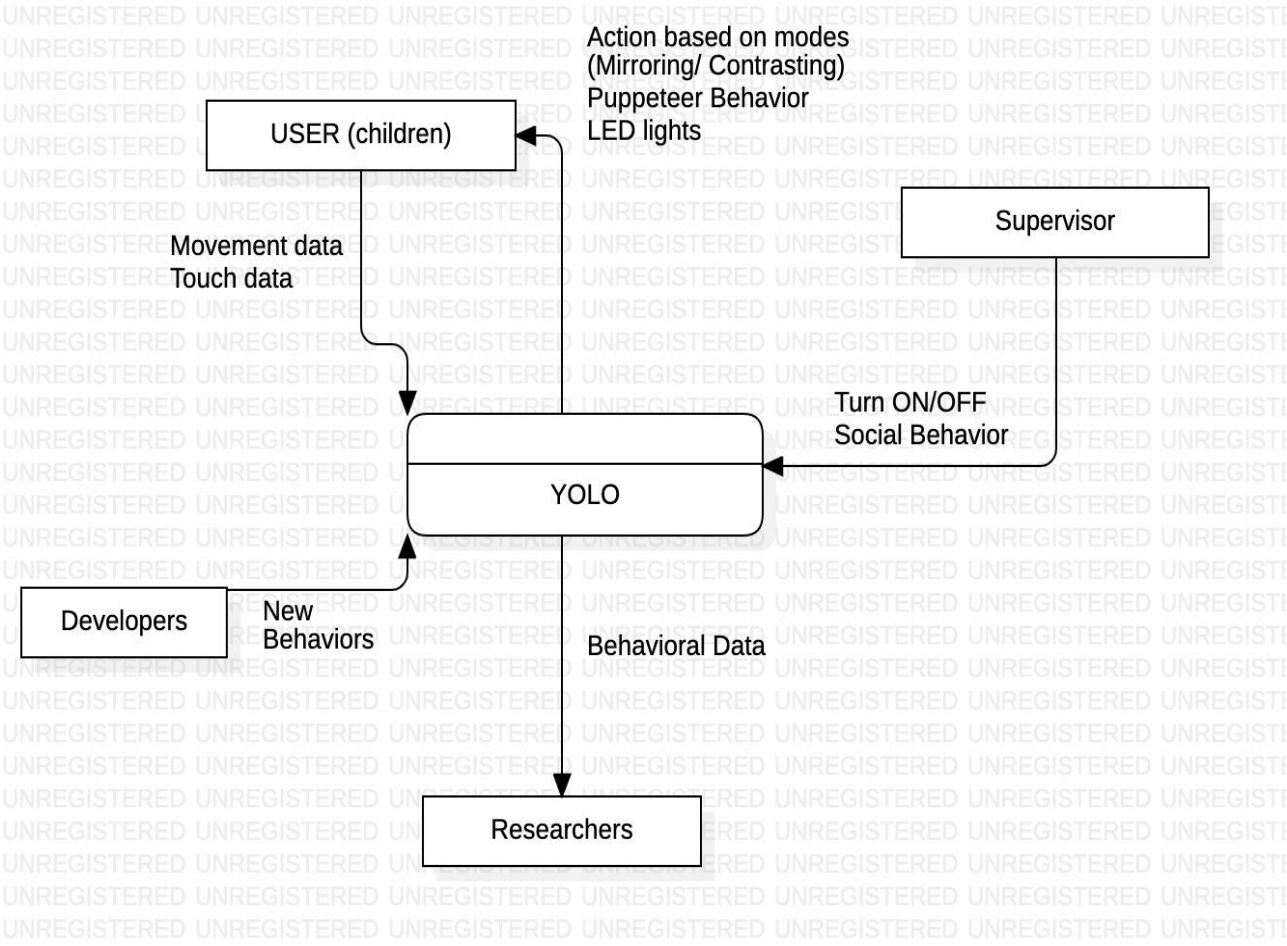

Figure 1: Context Diagram

As it can be observed from the context diagram, YOLO robot receives movement and touch data from the user, who is generally the child, so that it can process it later in order to function and carry out some movements itself. On the other hand, it is generally recommended that a supervisor such as a parent initializes (turn on/off) YOLO and sets the

social behavior of the robot to ensure that the robot acts according to the characteristics of the child. From a different perspective, researcher uses the YOLO robot in order to acquire behavioral data, which is used for psychological and social research that focuses on child-robot interactions. Development team and external developers are able to add new behaviors and features to YOLO robot.

4.1.3. External Interfaces

In this section, the external interfaces of the YOLO robot will be provided, as well as their operations and relationships between them and the robot and between themselves.

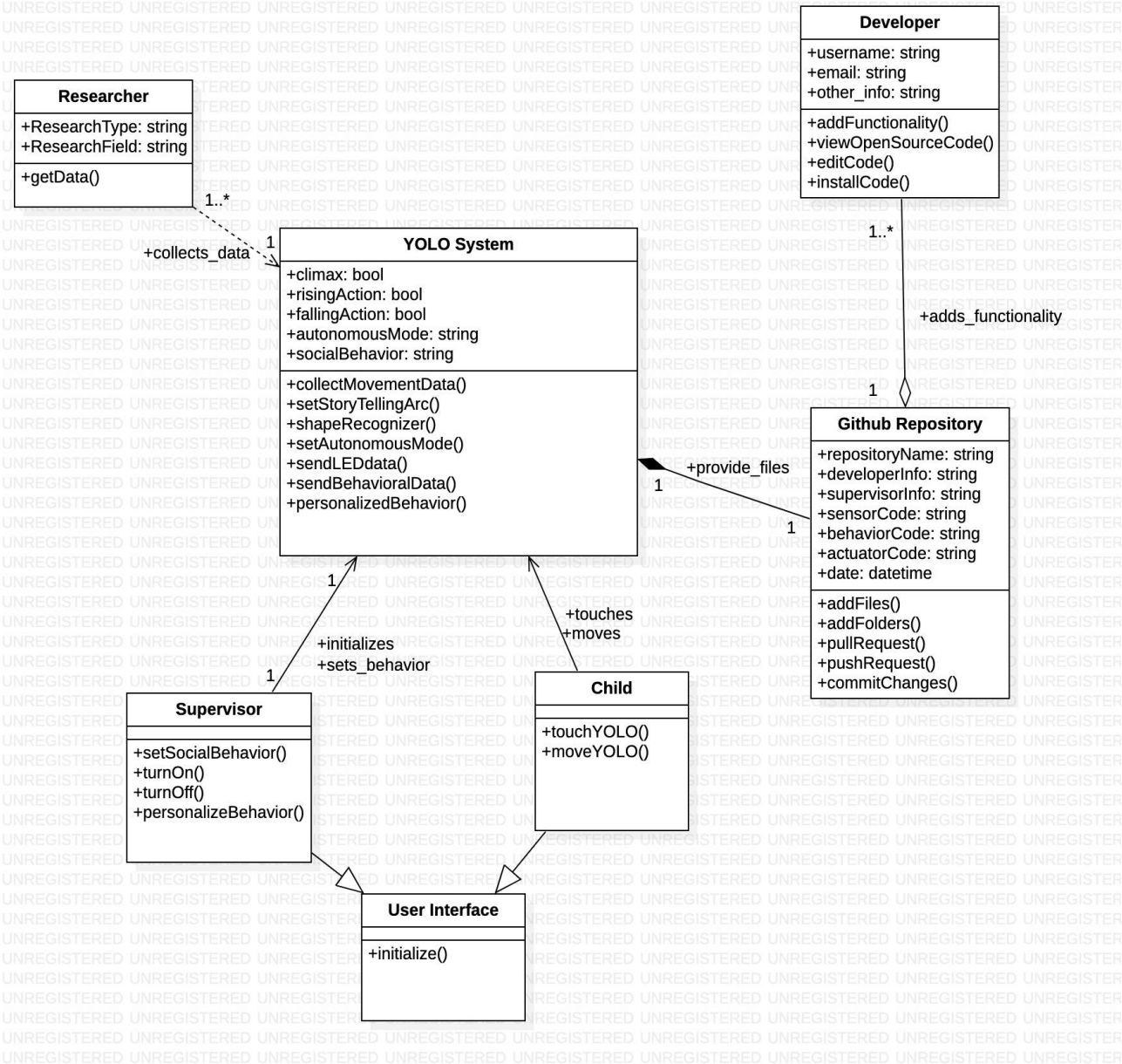

Figure 2: External Interfaces Class Diagram

As it can be observed from Figure 2, YOLO robot has multiple external interfaces. Researcher, Supervisor, Child and Developer are external stakeholders, and hence external interfaces of the system. Github Repository for the YOLO robot is also considered an external interface. The operations given in the diagram can be summarized as follows:

- The researcher collects behavioral data using the YOLO robot related to their field of study, which mainly covers psychological and social aspects of the child-robot interactions.
- The supervisor is responsible for initializing the robot, as well as setting the social behavior of it, based on the characteristics of the main user, the child.
- The child touches and moves the robot, allowing the robot to determine its actions based on the touch and movement information provided.
- Github Repository is used for almost all the functionalities of the robot. It provides necessary files, which include the source code for the YOLO robot's behavior, sensors, actuators, etc.
- Developer, although not directly related to the system, is able to view, install and edit the source code in the Github Repository. This way, they are able to add new functionality to the robot, since YOLO has an open source software.

4.1.4. Interaction scenarios

Two different interaction scenarios (one between child user and the YOLO System and the other between supervisor and the YOLO) are provided:

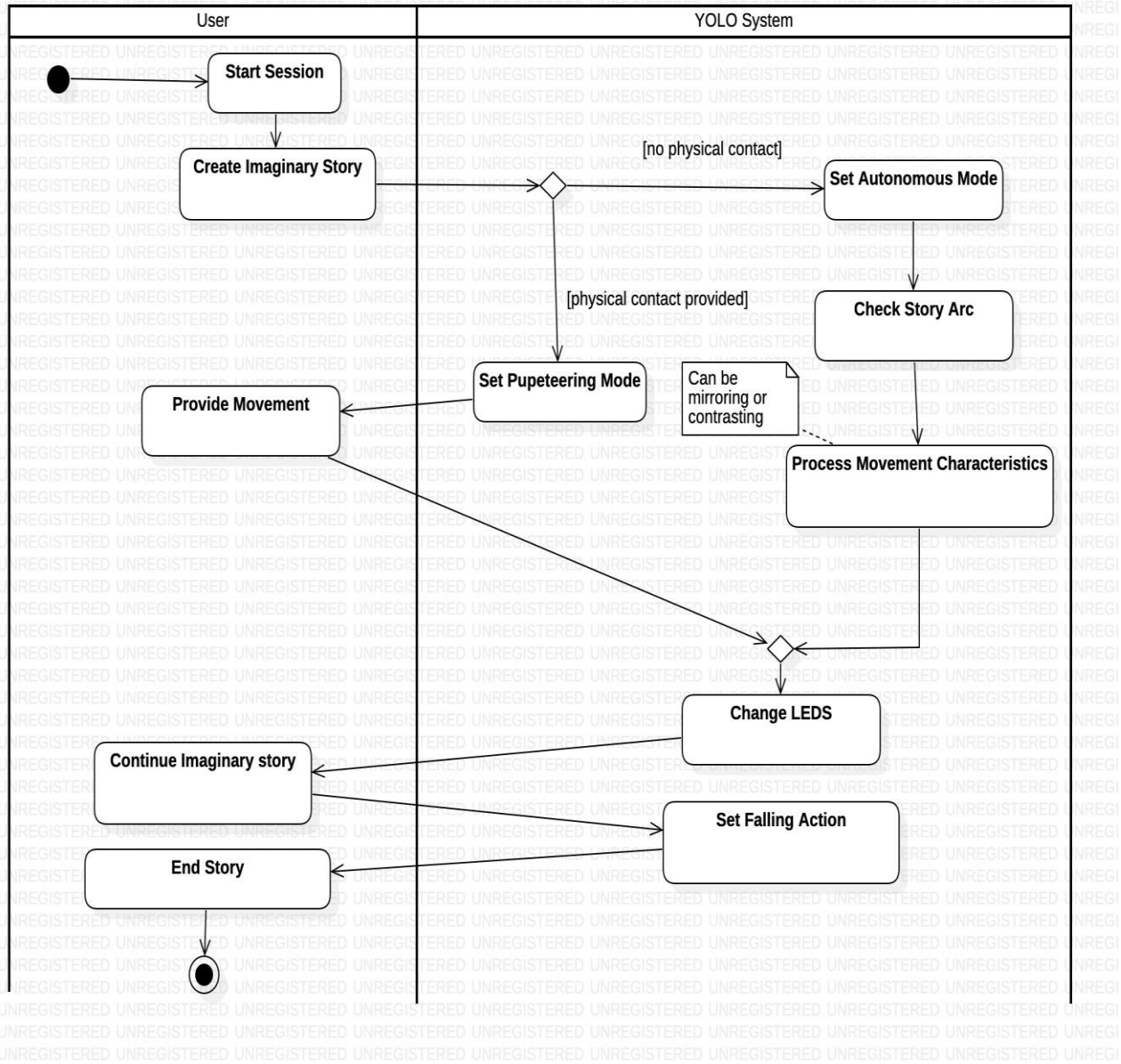

Figure 3: Activity Diagram for User-YOLO Interactions

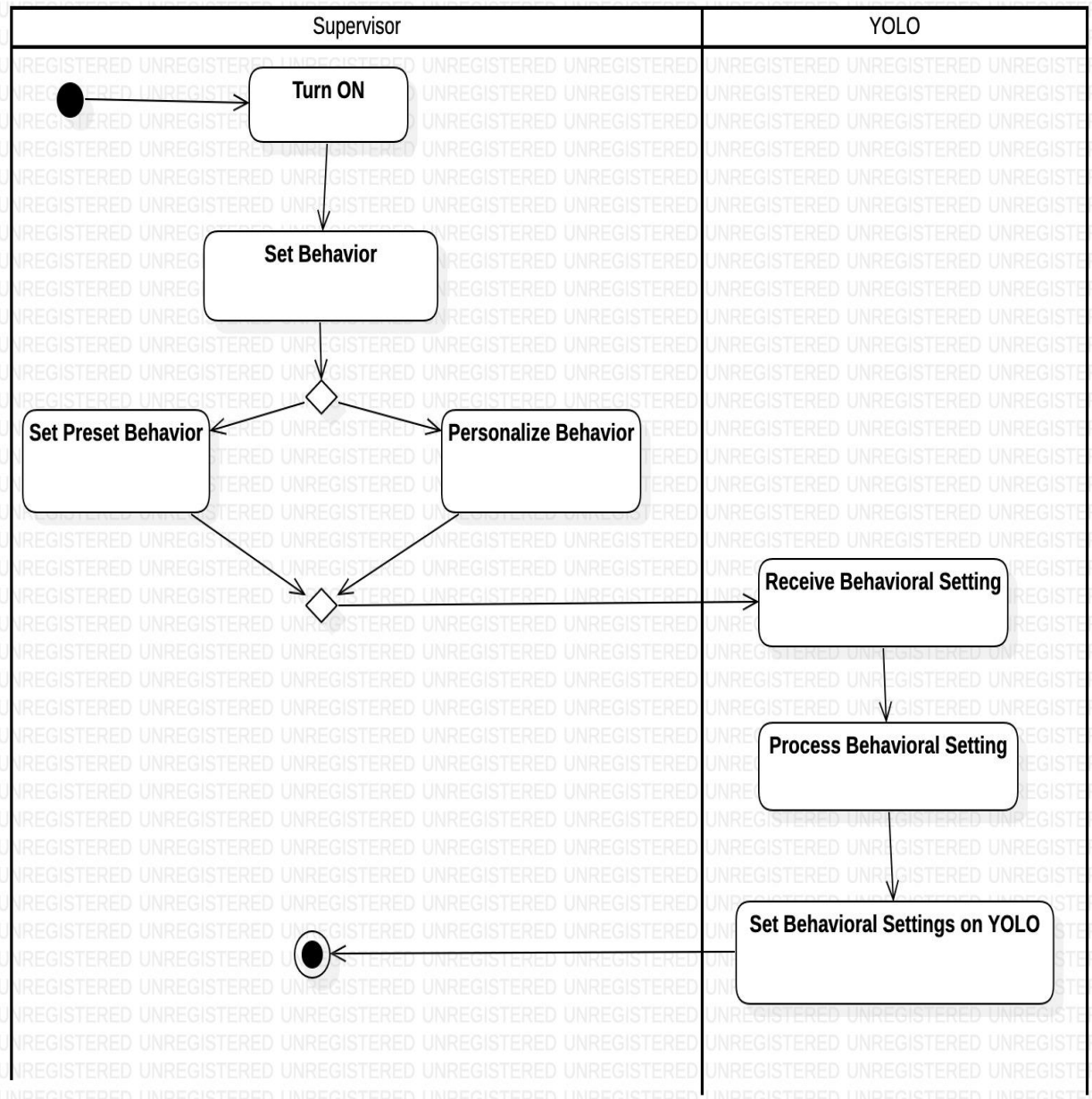

Figure 4: Activity Diagram for Supervisor-YOLO Interactions

4.2. Functional View

4.2.1. Stakeholders' use of this view

In this viewpoint, different components of the system, internal interfaces within the system, their responsibilities and interactions will be depicted. Functional view is extremely significant to the stakeholders of the YOLO robot, since it provides an idea of the quality properties and functionalities of the system and forms the shape of other viewpoints.

4.2.2. Component Diagram

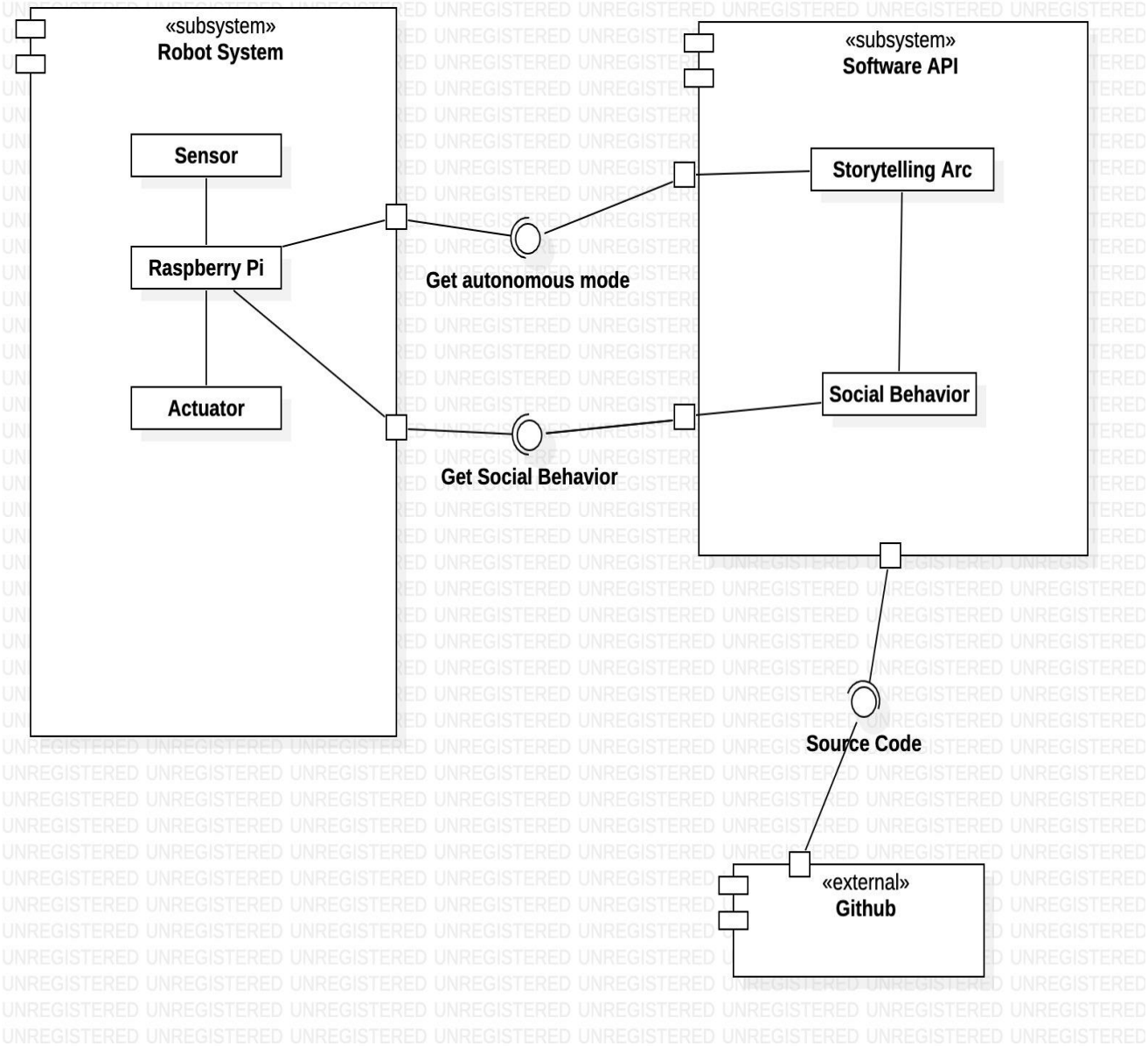

Figure 5: Component Diagram

Our system is divided into two main subsystems i.e. Robot System and the software API.

- GitHub is an external component which is the external interface required by the software API.
- The robot system is divided into components on the hardware of the robot such as actuators, sensors and the Raspberry Pi. The Raspberry Pi requires an interface to communicate key instructions which is provided by the Software API.
- To further elaborate, Robot system has a Sensor system which detects touch or movement vectors and relays it to the raspberry pi, which can either forward it directly to the actuator system or send it to the software API using an external interface for required processing of sensor data.
- The software system is further divided into two key components which include Story arc system which keeps track of the current story arc and provides an interface to robot system for setting the correct autonomous behavior (Contrasting or Mirroring). Secondly, it has a social behavior system where users can create personalize behaviors or choose from preset behaviors which provides the external interface required by the robot system to initialize the robot's behavior.
- This interface provided by software API to robot system completes two essential functionalities which are sending information about the state of autonomous behavior based on the story arc and social behavior that was preset in the software.
- The only external component in YOLO is GitHub which is the external interface providing the software API with source codes and the files which make up the software. Secondly, it can also be used to create new personalized behaviors in the software by developers.

4.2.3. Internal Interfaces

In this section, the internal interfaces of the YOLO robot will be provided, as well as the descriptions of their operations and relationships between them.

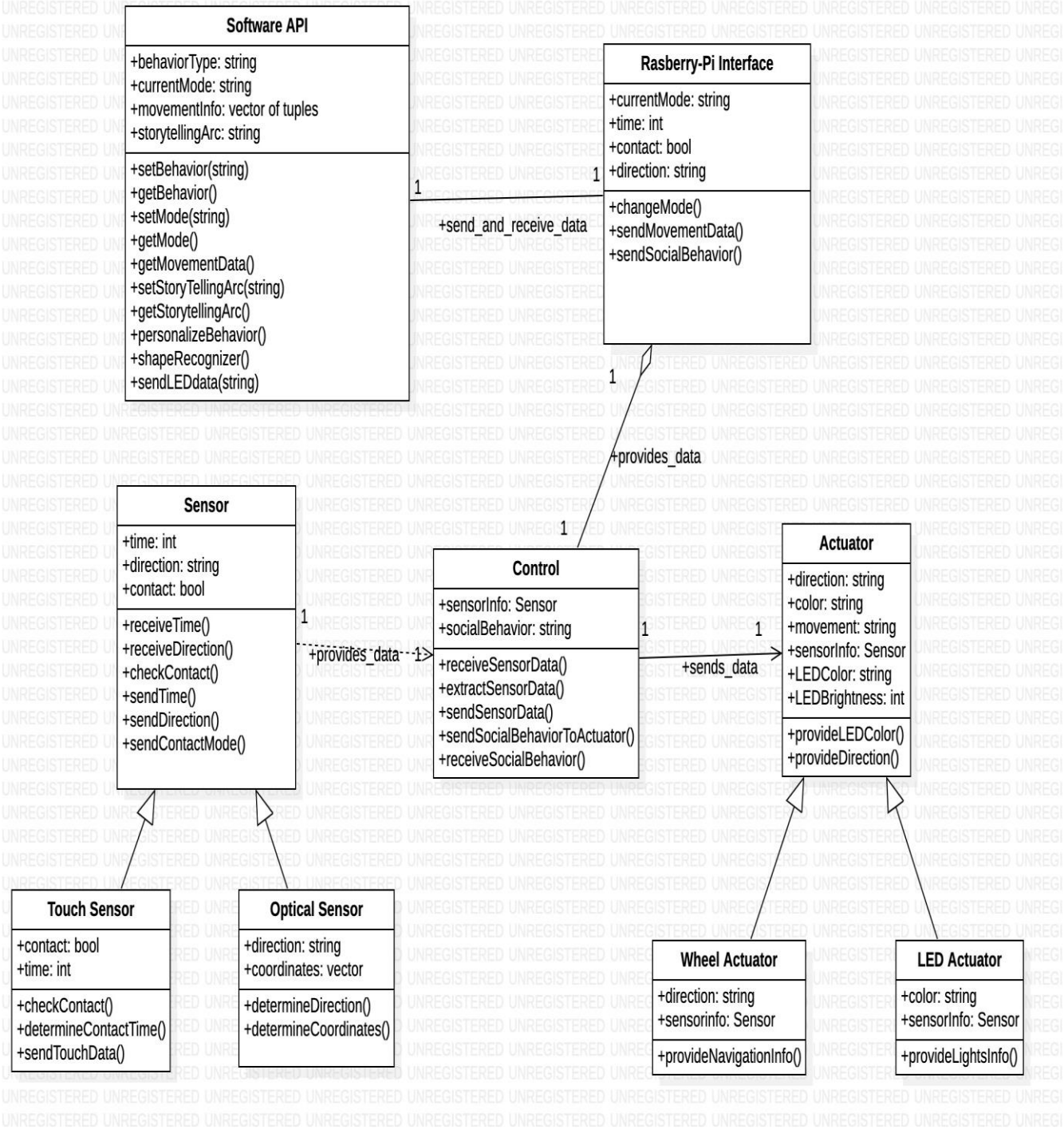

Figure 6: Internal Interfaces Class Diagram

As shown in Figure 6, the internal interfaces of YOLO are Software API, Raspberry-Pi, Sensor, Control and Actuator. The operations of each interface can be described in the following manner:

- Sensor Class has two subclasses, which are Touch Sensor and Optical Sensor. Touch sensor gets the information whether the robot is being touched by the user or not, whereas Optical sensor is used for detecting the movement direction. After collecting both data, it is provided to the Control interface;
- Control interface receives the data from Sensor interface and sends it to two other interfaces: Actuator and Raspberry-Pi. It has an aggregation relation with Raspberry-Pi, since Control can exist without that interface, but it is still dependent on it in providing communication with the Software API;
- Raspberry-Pi is directly connected to Software API interface and it sends necessary data, including sensor data to it to be processed by the API;
- Software API is responsible for setting important properties of the system, such as behavior, storytelling arc and current mode;
- On the different side, actuator gets the sensor information, as well as social behavior, which is then used to set LED properties, including color and brightness. Moreover, it provides navigation of the robot as well. It also has two subclasses, Wheel Actuator and LED Actuator, which are responsible for providing navigation and LED lights to the robot, respectively.

4.2.4. Interaction Patterns

The following diagrams represent messaging sequences taking place among the system components over the internal interfaces:

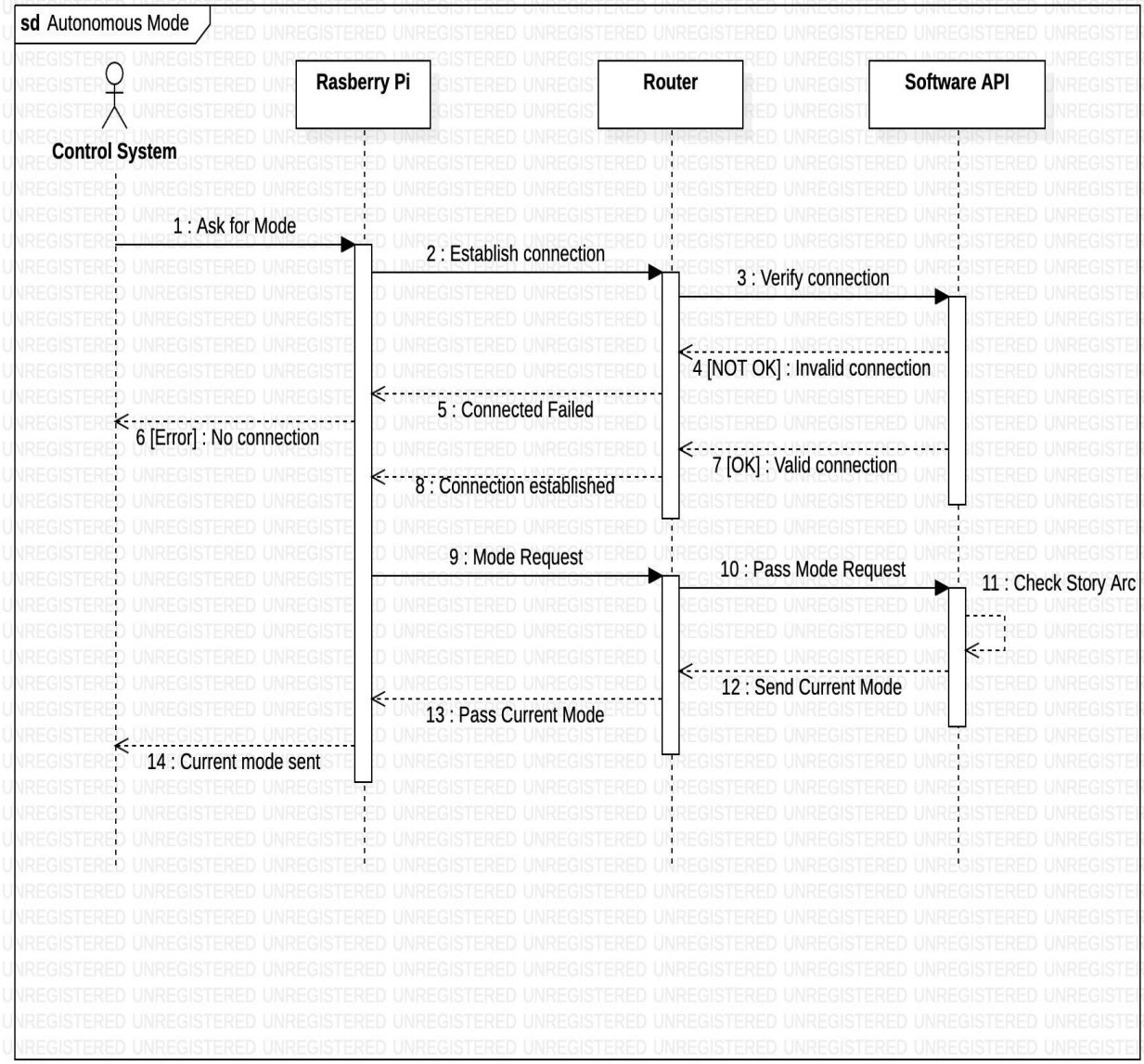

Figure 7: Sequence Diagram for Setting the Autonomous Mode

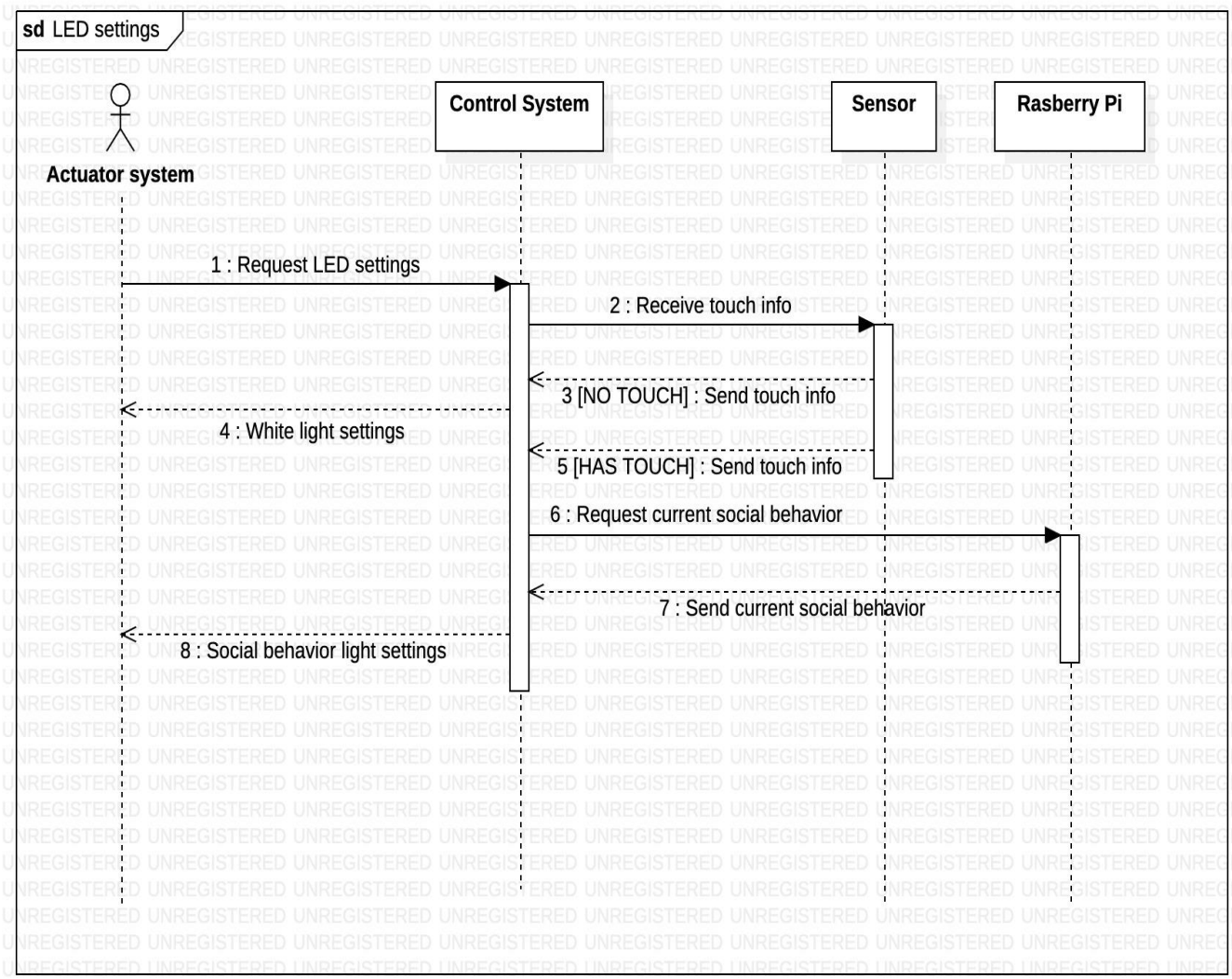

Figure 8: Sequence Diagram for Setting Up the LEDs

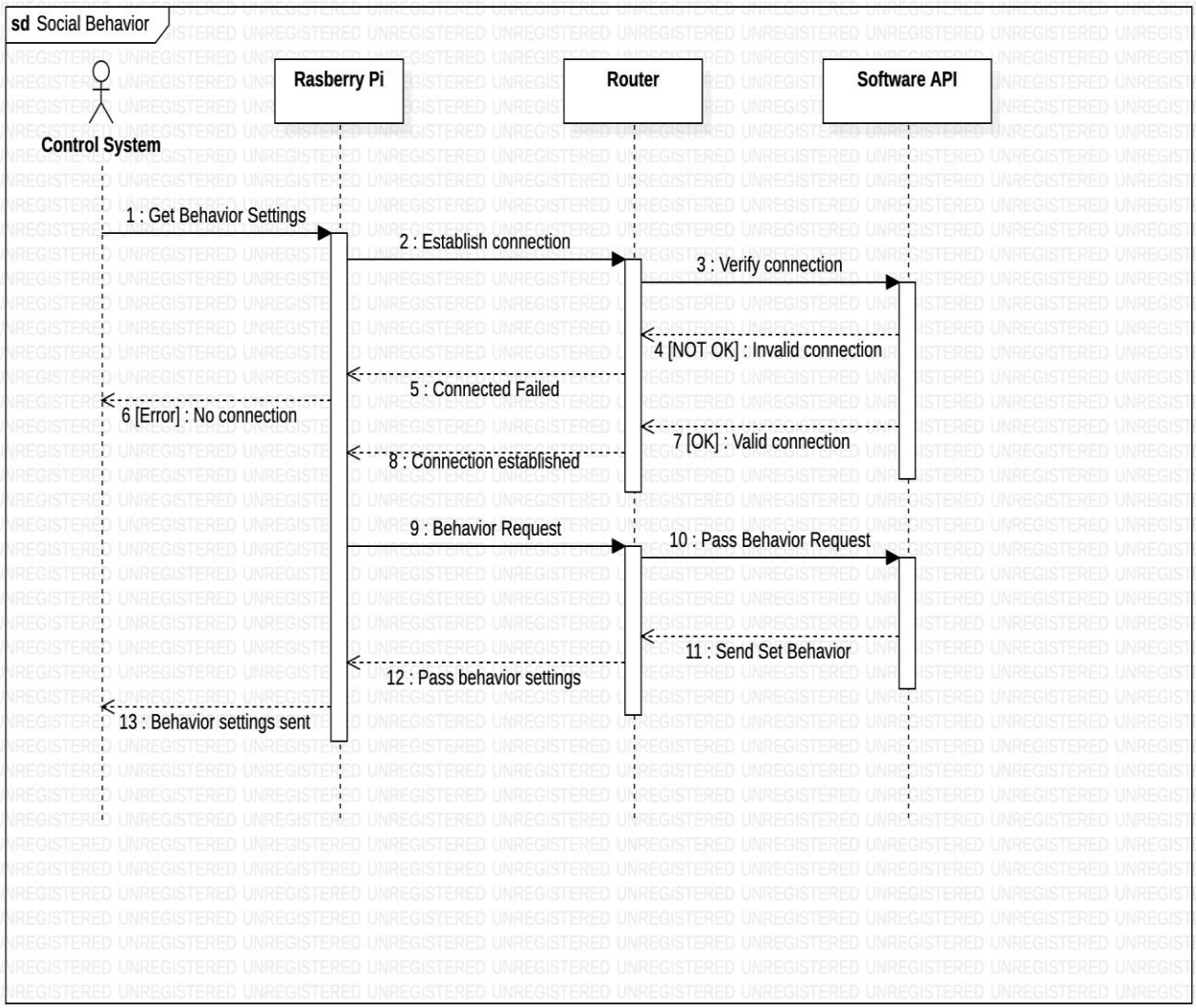

Figure 9: Sequence Diagram for Setting the Social Behavior

4.3. Information View

4.3.1. Stakeholders' use of this view

In this viewpoint, key database and main memory objects, as well as their associations will be provided. In that sense, it gives an idea of what kind of data flows occur, what kind of memory objects are used and how different data operations are carried out within the system. As a result, they have information about how data is stored, managed, manipulated and distributed in the system, which eases the use of different data objects and data manipulation techniques by the users.

4.3.2. Database Class Diagram

The following diagram represents the main memory objects and data stored in the system. No external database is provided, since YOLO uses an internal database.

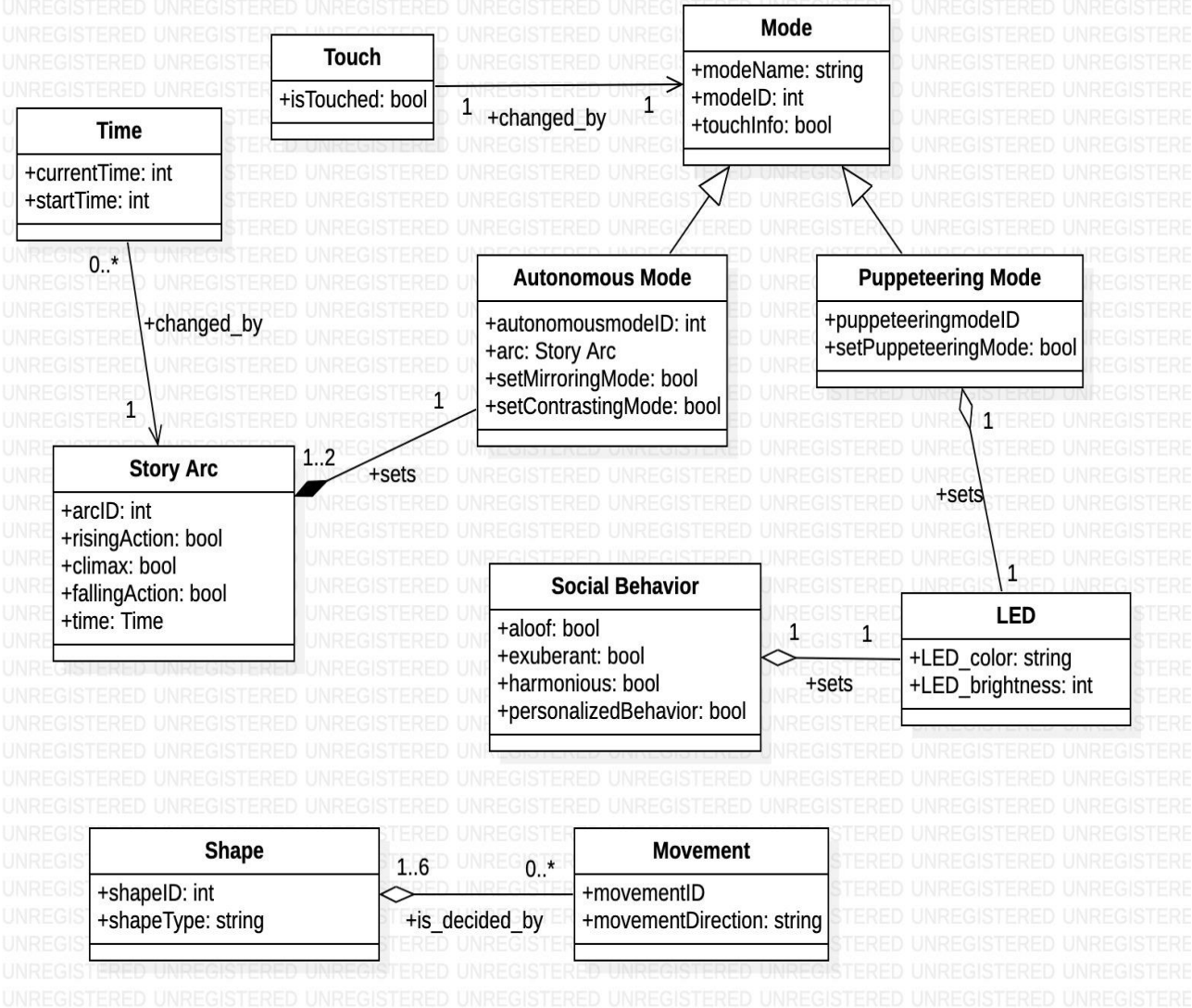

Figure 10: Database Class Diagram

The key database and main memory objects can be summed up using the following key points:

- Each social behavior (aloof, exuberant and harmonious) sets a certain color and brightness level for the LEDs;
- Puppeteering mode is associated with a specific color and brightness of LEDs;
- A specific shape may be used to decide several different movements, since there are only 6 different shapes available (circle, rectangle, curl, spike, straight line and loop). Similarly, a certain movement can be associated with more than one shape;
- As previously mentioned, there are 3 Storytelling Arcs: Rising action, Climax and Falling action; while Climax is associated with a single Autonomous Mode ("Contrasting"), both Rising and Falling action are associated with "Mirroring";
- Time is accessible to Story Arc, as its attributes are used to determine which Arc to set. Different times can be related to the same Arc. For instance, between 0h and 5h, Story Arc is set to Rising Action.
- Whether the user is touching the robot or not changes the Mode

4.3.3. Operations on Data

The following table represents the different operations that can be done on the memory objects given in the database class diagram. The operations are related to the storage and handling of different forms of information in the YOLO system:

Operation	**CRUD (Create/Read/Update/Delete)**
newPersonalizedBehavior	Create: Social Behavior, LED Read: - Update: - Delete: -
createMovement	Create: Movement Read: - Update: - Delete: -
createShape	Create: Shape Read: Movement Update: - Delete: -
updateSocialBehavior	Create: - Read: - Update: Social Behavior Delete: -

updateStoryArc	Create: - Read: Time Update: Story Arc Delete: -
setPuppeteeringMode	Create: - Read: - Update: Puppeteering Mode Delete: -
updateTouchSensor	Create: - Read: - Update: Touch, Puppeteering Mode Delete: -
updateAutonomousMode	Create: - Read: Story Arc Update: Autonomous Mode Delete: -
getShape	Create: - Read: Movement Update: Shape Delete: -
setMovement	Create: - Read: - Update: Movement Delete: -
updateLEDs	Create: - Read: Puppeteering Mode, Social Behavior Update: LED Delete: -
updateTime	Create: - Read: - Update: Time Delete: -
setMode	Create: - Read: Touch, Autonomous Mode, Puppeteering Mode Update: Mode Delete: -

Table 2: Operations on Data Table

4.4. Deployment View

4.4.1. Stakeholders' use of this view

Although this viewpoint might not be very useful for children, supervisors and researches, it is particularly useful for the developers, which are one of our stakeholders, since it describes the environment into which the system will be deployed, as well as the dependencies. It gives a better idea regarding runtime platforms, third-party software and network requirements, hosting of the software, etc., which are significant aspects after the software is built and is to be validation tested and put into real operation.

4.4.2. Deployment Diagram

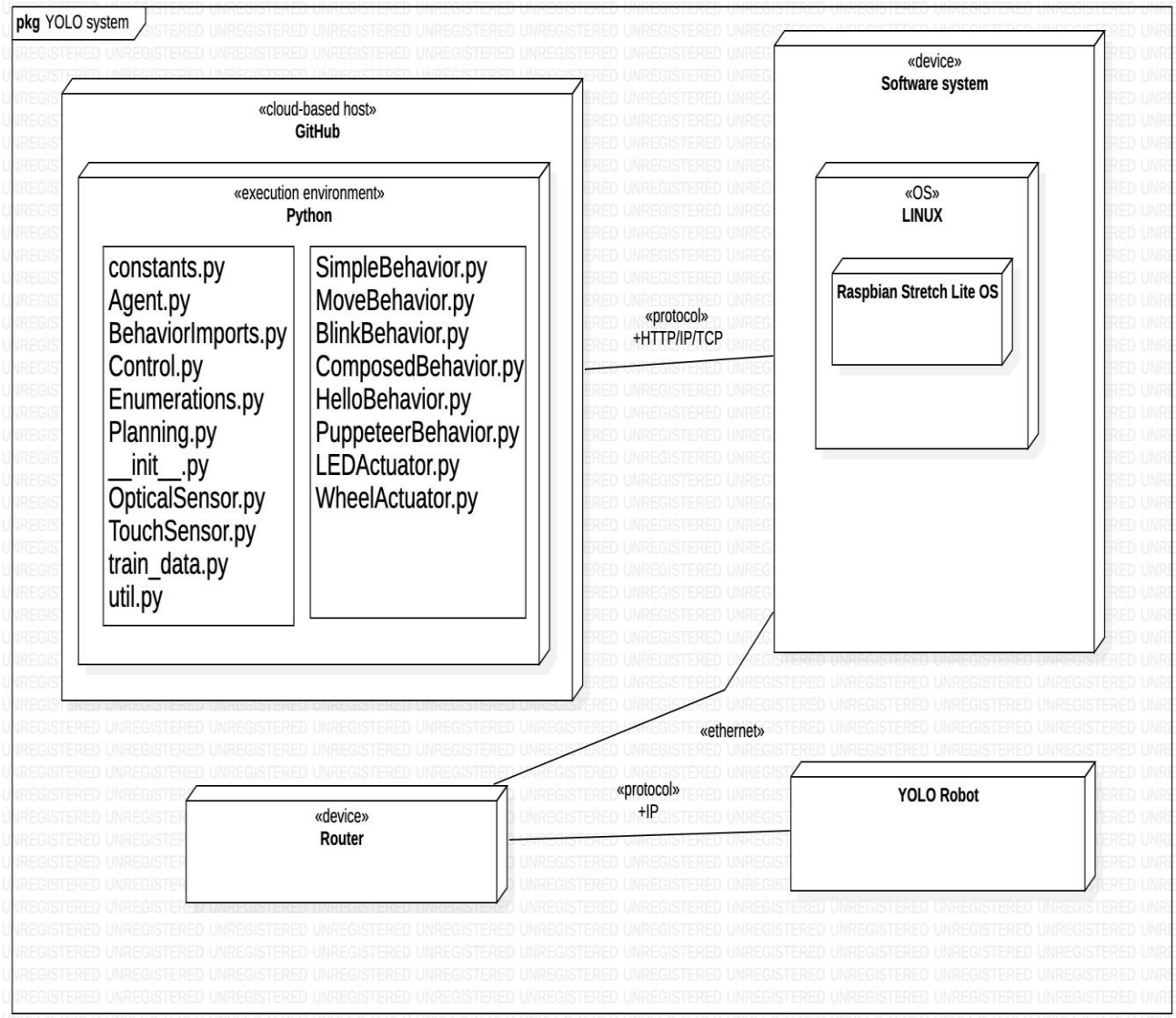

Figure 11: Deployment Diagram

In our deployment diagram viewpoint, we showcase the system's deployment environment and the dependencies that the system has on specific elements of it.
Important to note, that there is no external server required for a database since we don't make use of any. However, for our case and its open source nature, there is a code repository on GitHub which provides cloud-based hosting service. This computation system has our source files which use an execution system of Python, since our software is only coded using Python language. Inside of this execution environment we can see all the different artifacts provided, which are essentially almost exclusively .py or .txt files.

Our YOLO software system which comprises of a computer device communicates and downloads files from the GitHub using HTTP/IP/TCP protocols. It has certain dependencies on it such as the Computing platform being Linux with an installation dependency of Raspbian Stretch Lite OS. This YOLO system is also communicating with a router through an Ethernet or IP (Wi-Fi) protocols. There is a YOLO robot hardware, as well, which is essentially our playing robot, communicating with the a router through IP protocols

In this way, we have made clear all dependencies and laid out the specification and quantity of hardware devices required and the environment in which they will operate. Furthermore, all technological compatibilities as well as network and physical constraints were kept in check.

4.5. Design Rationale

i. Context View:

In the context view, a context diagram and a class diagram for external interfaces were used. The main purpose of the context view is to describe the relationships and interactions between the system and its environment. Therefore, in the system context diagram, 4 stakeholders (users, researchers, developers and supervisors) and their interactions with the YOLO robot were shown. Using the context diagram made the interactions more straightforward for the stakeholders to comprehend. Moreover, external interfaces diagram allowed us to exhibit different attributes (such as username, email attributes of the developer) and methods (such as setSocialBehavior, turnOn/Off, personalizeBehavior of the supervisor) related to each external interfaces and give more details about the interactions between them. We also included Github as one of our external interfaces, because it is one of the main interfaces in our system, which is used to provide files including the source code to the general system. Activity diagrams were also helpful in clarifying different scenarios, which might have been difficult to understand only using the class diagram. So, the usage of context, class and activity diagrams allowed a broader and more detailed context view of the system.

ii. Functional View:

The main design choice made was the usage of component and class diagram (internal interfaces) in the functional view. Functional view in general shows the runtime functional elements of the system and their responsibilities. In this sense, the elements were mainly shown in the component diagram, which included the robot system, software API and Github. Moreover, using class diagram for internal interfaces, just like in the context view, allowed for a more detailed description of these interactions and responsibilities. For instance, how the Sensor and Actuator are related to the Software API through Control and Raspberry-Pi interfaces became more clear with the class diagram. In order to further enforce the purpose of the functional view and make it crystal clear to the users and other stakeholders, three sequence diagrams were provided, which represented setting the autonomous mode, LEDs and social behavior in an order.

iii. Information View:

The database class diagram provided in this viewpoint shed a light on different database and main memory objects. Since there is no specific external database of the YOLO robot and the data is stored and manipulated within the internal database of the system itself, the class objects provided in the diagram were mainly main memory objects, including both persistent and non-persistent data within the system. The relations between different objects were also clearly shown using different elements of class diagrams of UML. As an example, movement had an aggregate relationship with shape, since movement could still exist without a certain shape, yet for it to be converted to useful data for our system, it has to be associated with one of the six aforementioned shapes. In addition, different operations that could be carried were listed to further illuminate how the data is used and manipulated.

iv. Deployment View:

The main rationale behind the deployment view is to describe the environment into which the system will be deployed and the dependencies of the system. For that purpose, a deployment diagram was used as the main tool. As previously mentioned, this way, the environments were portrayed more smoothly. The deployment view is the main concern of the developers, especially in our context, since YOLO is an open source software with a Github repository that includes all the necessary files related to different system components. With the deployment diagram, it is visualized how software system gets the files from Github using the necessary protocols (HTTP/IP/TCP), considering that it is a cloud-based host. Furthermore, how the robot hardware and software are related to the router using IP and Ethernet is shown. The deployment view and diagram show how simple the YOLO system to be changed to the developers and easy to set up and use to the users (in particular, supervisors).