Fetch Bot Final Report

Term: Spring 2018

Professor: Taehyung Wang

Class: COMP 380 Section 15888

Date submitted: May 18,2018

Group 1 Members:

- Christian Shadd
- Maria Verna Aquino
- Thanh Vu
- Giovanni Orozco
- Joseph Damian

Table of Contents

Project Description

(a) Project Overview

The project makes use of the different software design and engineering principles learned from COMP 380 in order to make an autonomous bot that is capable of recognizing images and using path finding algorithms to find a path to its target. The bot makes use of neural network libraries to recognize objects and makes use of the raspberry pi and arduino uno as its processors for image recognition, path finding and movement.

Main (Includes API Docs): https://cshadd.github.io/fetch-bot/
GitHub: (Includes API Code): https://github.com/cshadd/fetch-bot/
License: https://github.com/cshadd/fetch-bot/blob/master/LICENSE

(b) Purpose of the Project

To create a robot that will demonstrate the use of artificial intelligence through image processing, pathfinding, and feedback mechanisms.

(c) Scope of the Work

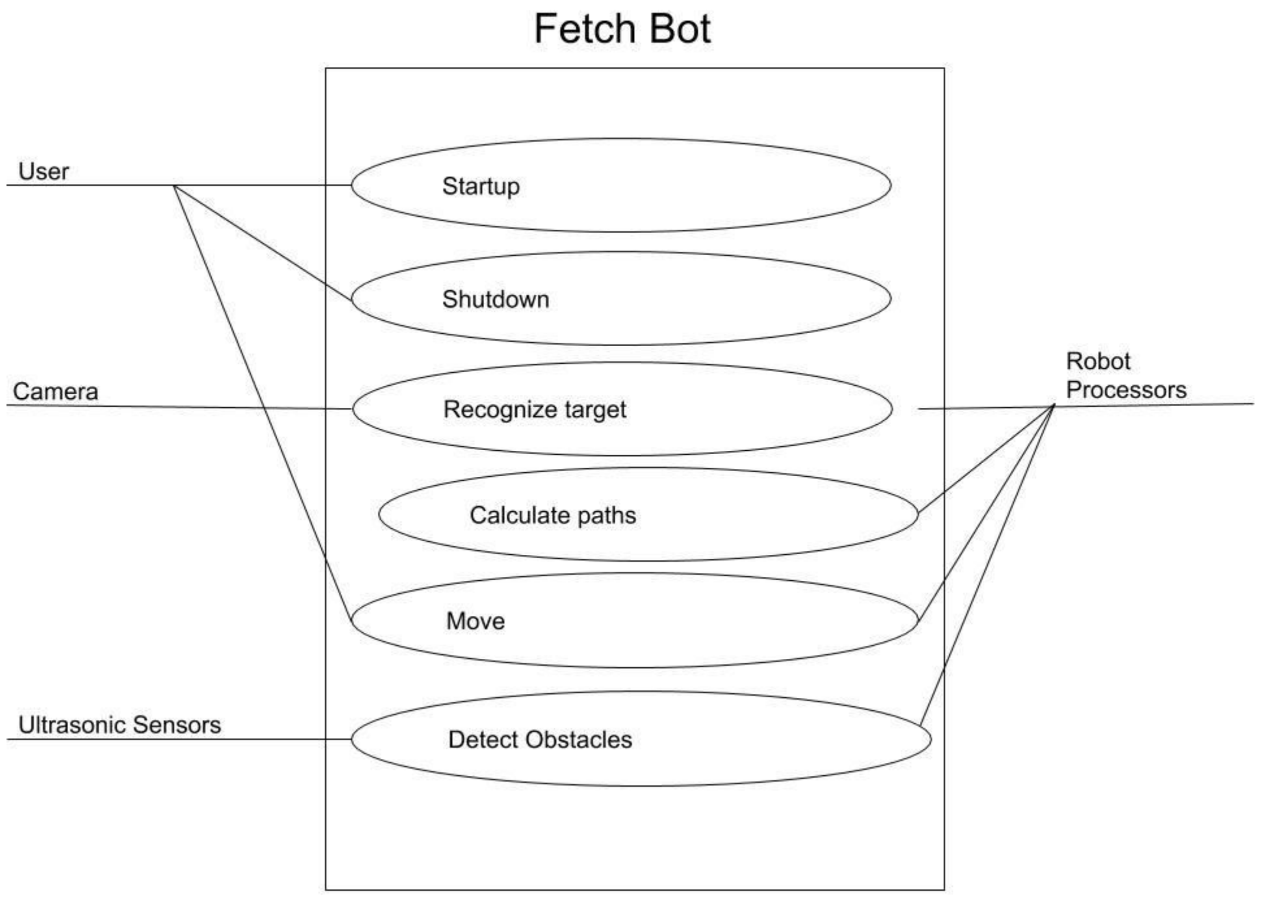
This project will consist of creating an autonomous bot that can detect a target and navigate simple obstacles until it reaches the target. The pre-alpha version Cinnamon has been completed as of May 2018. The AI robot can be controlled remotely via a web interface as well as autonomously by inputting a target to find in the web interface.

(d) Stakeholders

- Professor Taehyung Wang
- Group members
- Classmates

Requirements and Specification

(a) Use Case Diagram and Descriptions



Start up

- the robot is started up when the user runs a bash script.

Scenario	Start up
Triggering Event	User starts the robot up.
Actors	User
Related Use Cases	N/A
Stakeholders	User
Pre-condition	Robot Processors is OFF.
Post-condition	Robot Processors is turned ON.

Flow of Events

Actor	**System**
1. User turns power on. 2. User starts bash script.	3. Preform startup diagnostics. 4. Check repo update and compare software version. 5. Open interface face.

Exception

Step	**Condition**	**Action Desc**
3a	Startup diagnostics failure	Save to log file and notify user.
4a	Repo update check and software version comparison failure	Save to log file and notify user.
5a	Interface face open failure.	Notify user.

Shutdown

- the robot shuts down when the user clicks the kill button on the web interface.

Scenario	Shutdown
Triggering Event	User shuts the robot down.
Actors	User
Related Use Cases	N/A
Stakeholders	User
Pre-condition	Robot Processors must be ON.
Post-condition	Robot Processors is turned OFF.

Flow of Events

Actor	**System**
1. Press KILL switch on web interface.	2. Closes all threads and robot tasks.

	(nested)

	3. Preforms safe shutdown.

Exception

Ste p	Condition	Action Desc
2a	Thread and task closure failure.	Save to log file and notify user. Skip 3 and perform hard shutdown.
3a	Safe shutdown failure.	System will crash to prevent damage.

Recognize Target

- the robot detects target using its camera.

Scenario	Recognize Target
Triggering Event	Robot camera detects the target.
Actors	User, Camera, Robot Processors
Related Use Cases	N/A
Stakeholders	User
Pre-condition	Target in range of Camera.
Post-condition	Robot Processors detect a target.

Flow of Events

Actor	**System**
1. User trains image processing neural network for specified target.	2. Neural network will scan frames from webcam input. 3. Send confirmation if target is found.

Exception

Step	Condition	Action Desc
2a	Neural network failure.	Save to log file and notify user.

	3a	Confirmation failure.	Save to log file and notify user.

Calculate Paths

- the robot calculates a path. In autonomous mode, robot generates random paths until a target is found. Robot then calculates path to target.

Scenario	Calculate Paths
Triggering Event	Robot calculates paths.
Actors	User, Robot Processors
Related Use Cases	Detect Target
Stakeholders	User
Pre-condition	Robot Processors is in MANUAL mode.
Post-condition	Robot Processors calculates graph paths.
Flow of Events	

Actor	**System**
1. User sets mode to autonomous.	2. Calculate graph points. 3. Setup graph. 4. Use graph for movement.

Exception

Step	**Condition**	**Action Desc**
2a	Graph point calculation failure.	Save to log file and notify user.
3a	Graph setup failure.	Save to log file and notify user.

Detect Obstacles

- the robot detects obstacles in its path by using its ultrasonic sensors.

Scenario	Detect Obstacles

Triggering Event	Ultrasonic Sensors detect obstacle.
Actors	Ultrasonic Sensors
Related Use Cases	Calculate Path
Stakeholders	User
Pre-condition	Obstacle within Ultrasonic Sensors range.
Post-condition	Ultrasonic Sensors detect the obstacle.
Flow of Events	

Actor	**System**
1. Ultrasonic Sensors detect obstacle.	2. Serial data sent to Robot Processors for review.

Exception	

Step	**Condition**	**Action Desc**
2a	Serial IO failure.	Save to log file and notify user.

Move

- robot moves after receiving a user command, detecting a target/obstacle, or calculating a path.

Scenario	Move
Triggering Event	Robot Processors has received User command, detected target, calculated path, or detected obstacle.
Actors	Robot Processors
Related Use Cases	N/A
Stakeholders	User
Pre-condition	Data.
Post-condition	Robot Processor will sent movement command to motors.
Flow of Events	

	Actor / **System** 1. Robot Processors have data sent. / 2. Validation. 3. If validation checks out, Robot Processors will order motor to run.
Exception	

Actor	System
1. Robot Processors have data sent.	2. Validation. 3. If validation checks out, Robot Processors will order motor to run.

Exception

Step	Condition	Action Desc
2a	Validation failure.	Save to log file and notify user.
3a	Motor movement failure.	Save to log file and notify user.

(b) Functional Requirements (User and System Requirements)

I. The system shall be able to start up. During start up, the bash script, “bash/install.sh” is ran in the terminal. The bash script sets the monitor to never turn off. The script also kills any existing Java processes and starts the jar file which is the main program itself. It then opens up the chromium web browser on the monitor. It starts the program that keeps the mouse cursor turned off on the monitor to get the screen to only display the face of the robot.

II. The system shall be able to turn off. Once kill the button is pressed on the web interface, a log file is created. Serial ports are disconnected and the jar file is terminated.

III. The system must be able to recognize a target. In order to recognize a target, the robot is set to autonomous mode. Once the robot is set on autonomous mode by clicking the Auto button on the web interface, the robot uses the neural network on the raspberry pi to match images to preloaded data. It processes the image and if a 90% match is obtained, the robot reports the image as a target. If any error occurs for the neural network or the target being found, the error is saved to a log file and the user is notified.

IV. The system must be able to calculate a path towards the target. In order to calculate the path to the target, the robot is set to Auto mode. During auto mode, if the target is not found, it chooses where to go next and generates a graph as it moves around. Once the target is found, a straight path is generated towards it and is followed. If an obstacle is detected, another path is calculated and the graph is updated. This repeats until the target is reached. If any error occurs during the calculation and setup, the error is saved to a log file and the user is notified.

V. The system must be able to move. The robot can move in both Manual and Auto mode. In order to move during Manual mode, the user can choose to press the move buttons: Forward, Left and Right in the web interface in order to move the robot. In order to move in Auto mode, the user only needs to specify a target from a list of pretrained data in the interface. If the motors fail or the processor was not able to receive data, the error is saved to a log file and the user is notified.

VI. The system must be able to detect obstacles. Data is received from the sensors and the Arduino uno sends this to the raspberry pi via Serial which then gets sent to the jar file in the raspberry pi. The raspberry pi processes the data and interprets it into distances that will be sent to the web interface.

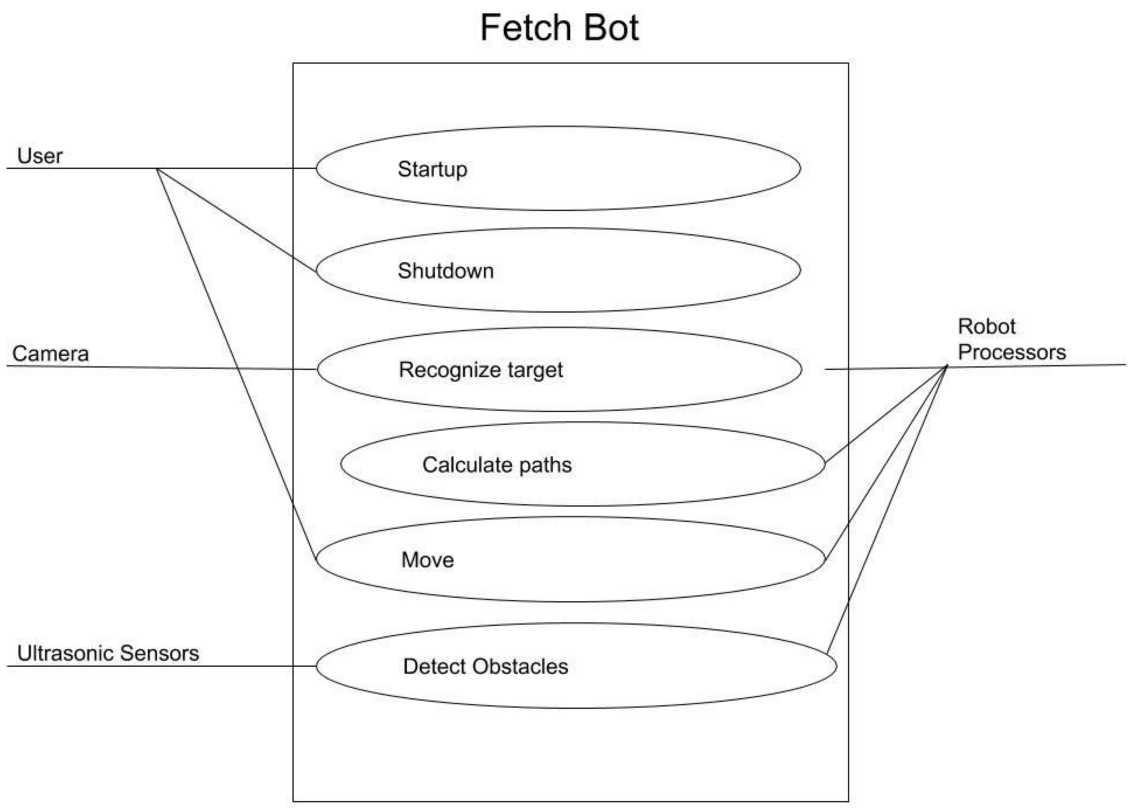
(c) Non-functional Requirements

I. Robot should have protections against outside intrusions.
II. Robot HTTP Server should have protections against outside intrusions.
III. Robot should receive a command within 3 seconds.
IV. Robot should process image data to at least a 90% match before a target is identified.
V. Robot should stop and turn when an obstacle is detected 15-30 cm away.

Design

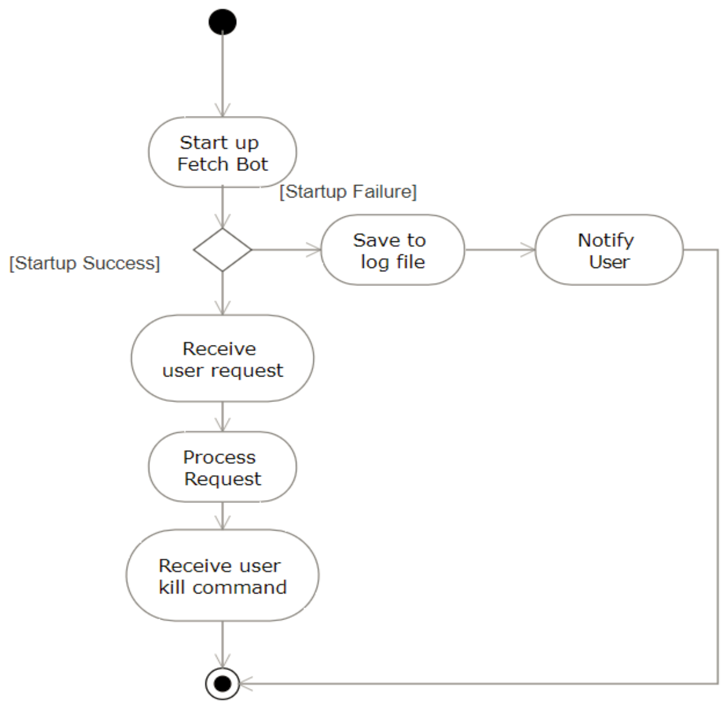
(a) System Modeling (five system models)

I. Context Modeling

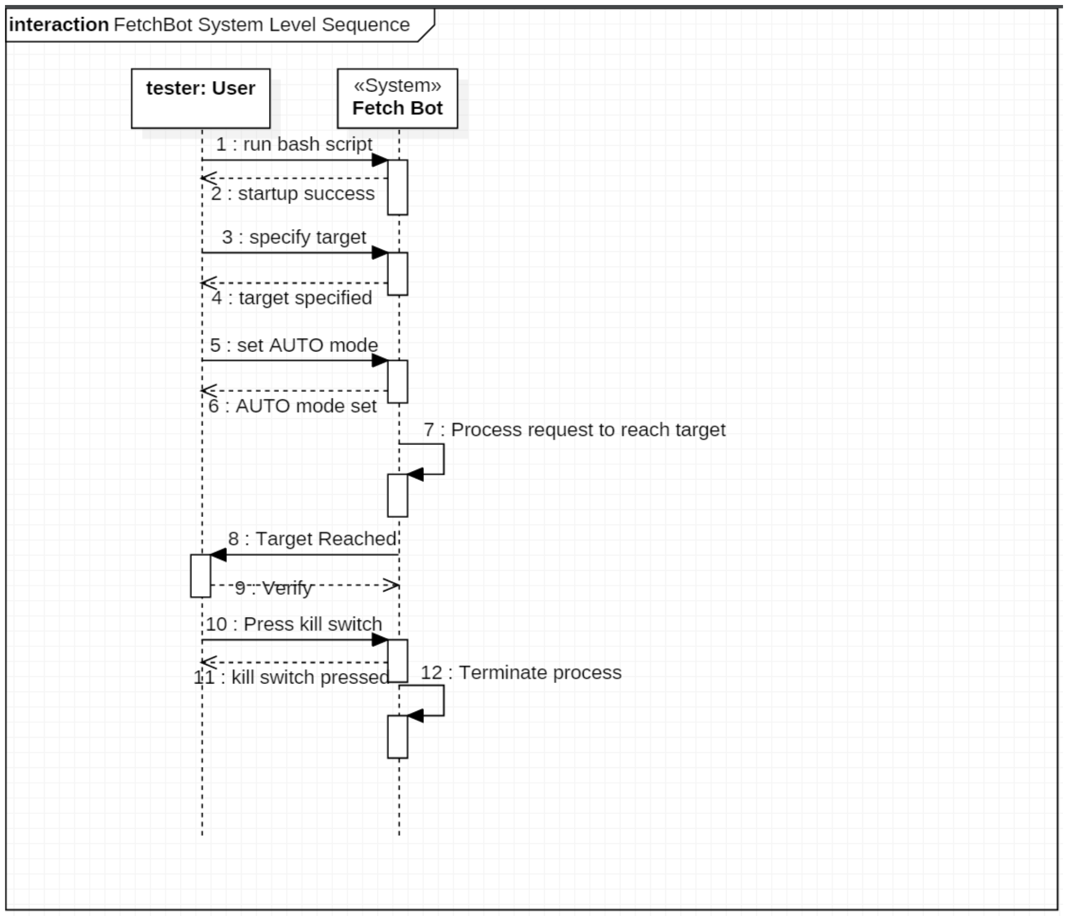


II. Process Modeling

Fetch Bot System Level Activity Diagram

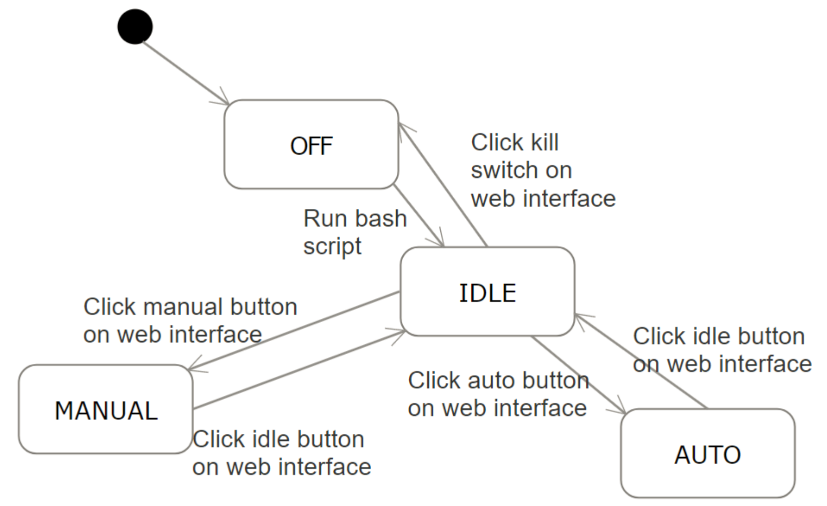


III. Interaction Modeling

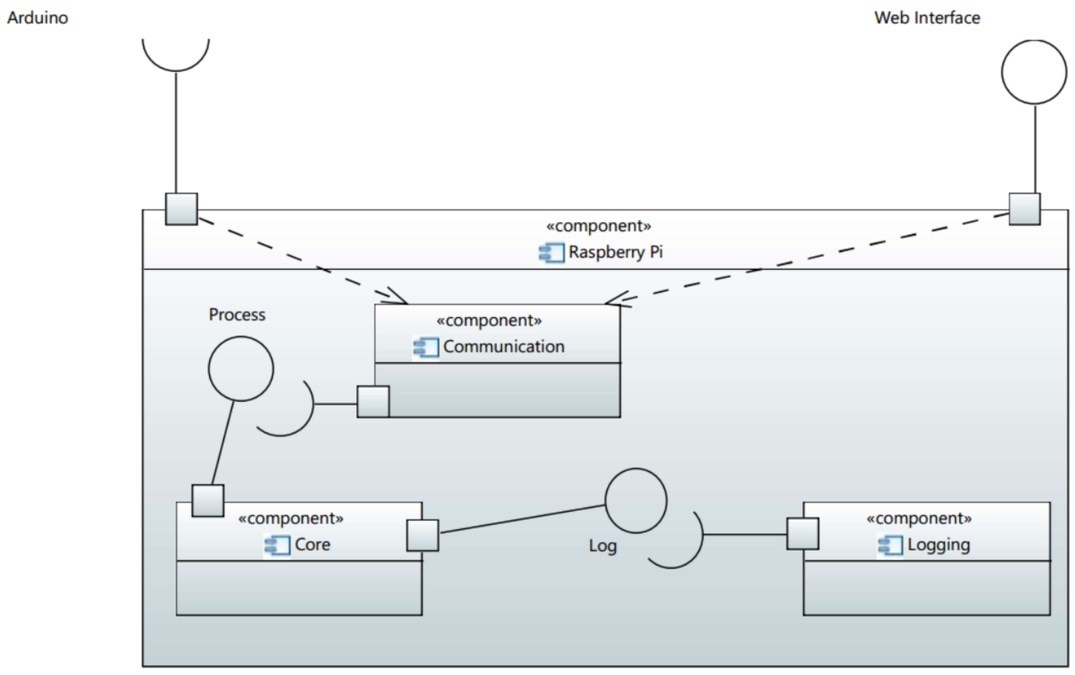


IV. Behavioral Modeling

Fetch Bot System Level State-chart Diagram


OFF
Click kill switch on web interface
Run bash script
IDLE
Click manual button on web interface
Click idle button on web interface
Click auto button on web interface
MANUAL
Click idle button on web interface
AUTO

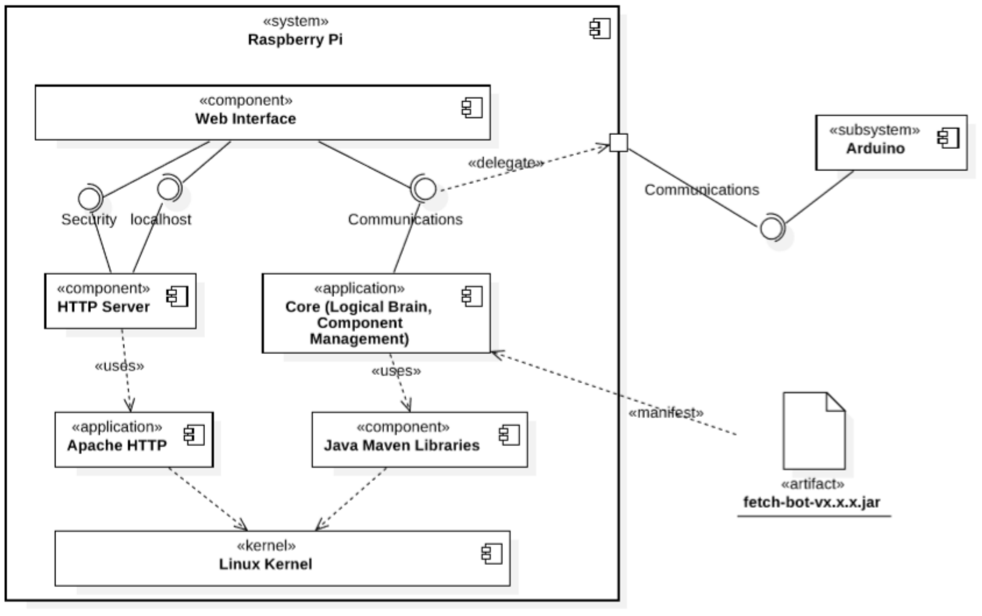
V. Structural Modeling



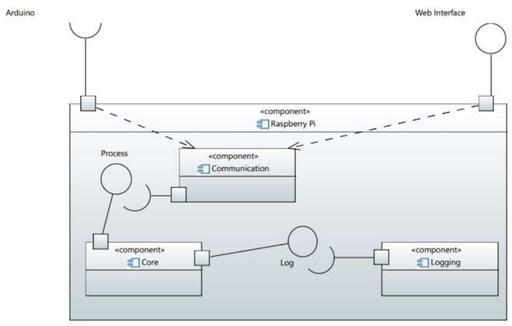
(b) System Architecture and Patterns

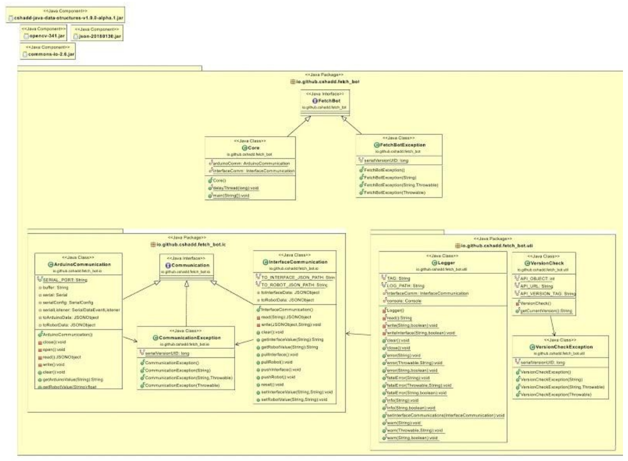
The software system for Fetch Bot is hierarchical as the backend uses components compiled directly from the kernel and uses drivers from the kernel itself. Image processing occurs within the kernel itself as OpenCV uses libraries (.so). Serial and I/O is on the backend. There is also the Java Virtual Machine running on the front end with a web interface.

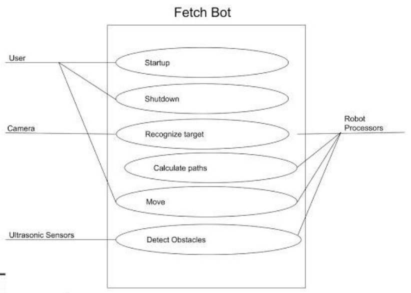
Fetch Bot Layered Architecture Pattern

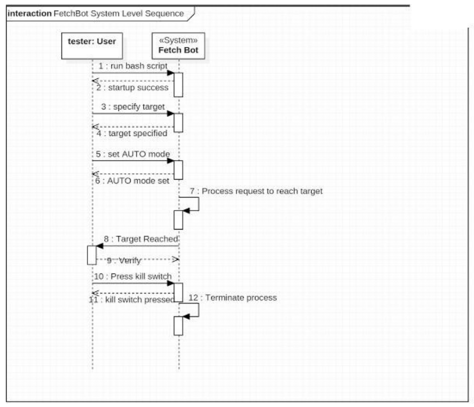

«system»
Raspberry Pi
«component»
Web Interface
Security
localhost
Communications
«delegate»
Communications
«subsystem»
Arduino
«component»
HTTP Server
«application»
Core (Logical Brain, Component Management)
«uses»
«uses»
«manifest»
«application»
Apache HTTP
«component»
Java Maven Libraries
«artifact»
fetch-bot-vx.x.x.jar
«kernel»
Linux Kernel

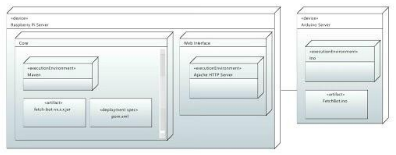
(c) 4+1 views (using UML Diagrams)











(d) Detailed design principles (five principles)

- The open–closed principle (OCP)
  Most classes are able to be extended when needed without having to modify existing code.

- The Liskov substitution principle (LSP)
  Base classes were used that extended from other classes when needed.

- The interface segregation principle (ISP)
  Base and child interfaces were used that encompassed their own needed

methods. If we needed more methods we extended them from other interfaces that made sense to extend from.

- Dependency Inversion Principle (DIP)
  Modules were kept separate and only called upon other modules when needed. Low level modules went through other low level modules in a stair-like fashion before being called by a high level module. Basically there were no skips between module levels.

- Single Responsibility Principle (SRP)
  Each class has its own dedicated job as documented, no two classes do the same job. Classes were also separated by packages or application in what they need to do.

(e) Detailed design patterns

We used some aspects of the Behavioural Design Pattern.

Testing

(a) Non-execution testing (walk-through and inspection)

Periodic software reviews were done to make sure that the various algorithms (e.g. Pathfinding) made sense. The group brainstormed and identified potential problems in the algorithms. For example, the appropriate distance necessary to turn away from an obstacle once detected was discovered during an informal walkthrough.

(b) Execution testing (black box testing)

Constant evaluation from executions were made to test the robot. There was a debug mode that allowed for verbosity. There was a utility logger that allowed logging to a file including debug messages, warning messages, info messages, and error messages thrown by errors or exceptions. The messages were thoroughly detailed and included stack trace which we used to validate the errors.

Project/Process

(a) Open issues

- Making the OpenCV terminal faster
- Adding GPS feature
- Adding encoders for movement precision
- Creating a full chassis
- Fixing latency

(b) Project/Process retrospective

The SCRUM process could have been modified to account for different availabilities of team members. A lighter battery could have been used in order to increase torque and allow the bot to move faster. Alternatively, stronger motors could have been used. The omni wheels were not necessary since only two motors were used and left-right movement would not have been possible. The use of regular front wheels could have decreased the project cost.

Copyright