**Term Project : Drone Butler**

**By : Maya Filipp, Aaron Patterson, Marko Radmanovic**

**BTR490 : Investigative Research Internship**

**To : Barb Czegel**

**Submitted : June 17 2016**

**Contents**

BTR490 System 2016

(1) System Name and Team Number

(2) Team Members

(3) Client and/or Mentor

(4) System Overview

(5) Technology to be used

(6) Stakeholders

(7) A Diagrammatic Representation of the System

(8) System Components

Raspberry Pi 2

Arduino MC

Main Interface

Drone movement

Flight movement

Flipping and Spinning

Drone Rotating/Tilting

Drone Landing/Takeoff

Drone Camera

Record Video

Video settings

Capturing Screenshots

Screenshot Settings

Recording Log Files

Status Menu

Moving objects

Picking up objects

Pushing objects

Ardupilot

(9) Who Did What Log

# **BTR490 System 2016**

## **(1)** **System Name and Team Number**

SkyNet (Sky – Drone, Net – Network communication, SkyNet – The future is now)

Team Number 14

## **(2)** **Team Members**

Maya Filipp

Aaron Patterson

Marko Radmanovic

## **(3)** **Client and/or Mentor**

Client: Ourselves

Mentor: Professor John Blair (Greg) of the School of ICT

## **(4)** **System Overview**

The system we are building is both a working (flying) Drone and a programmed interface that we use to communicate with it. We are aiming to have it stabilize itself, be able to change directions, be able to flip and do basic commands. We also hope to have the video feed working well and that it can go to a relatively good distance from the interface and still be able to connect with it.

Once the system is sufficiently working in the basic manner of flying without crashing, we hope to expand to include tests with pushing objects, pulling objects, carrying objects and other maneuvers that we come up with.

## **(5)** **Technology to be used**

Drone:

- We are going to use a Raspberry Pi 2 to power the brain of the drone.

- We will be looking into an Arduino microcontroller (still considering what we need to get because they come in various sizes – we need to minimize load on the drone) to control ongoing flight.

- The microcontroller must be compatible with ArduPilot - it is an open source platform that we will most likely be using – it is programmed to work best with robots that fly. We’re considering the ST Mirro ST32F3 Discovery Board, which our mentor mentioned being the grandfather of the ardupilot compatible boards.

- The microcontroller will also have to be an Rtos (real-time operating system) so that it can continually manage flying the drone while it receives instructions from the Pi (this is done so that it doesn’t stop controlling the propeller motors as it’s getting instructions from the pi, and subsequently crashes to the ground).

- We will also be looking into various batteries (most likely LiPo Battery (lithium ion polymer battery), but there are dangers with working with them). Again, the main obstacle is dealing with the amount of weight the Drone will have to fly.

- On top of Lipo batteries, we will be using a lot of Low current coin cell (CR2032) batteries to power the motors for the propellers.

- Speaking of motors, we will need 4 motors (quadcopter - 1 motor per propeller), and it will most likely be Brushless DC motor (3/4 wire to properly connect to the microcontroller). These motors are fast and light.

- Regarding sensors such as gyroscopes, accelerometers, compasses and even gps, we will be looking into the IMU Pololy, which is an electronic board that has all these sensors imbedded into it. Again, these components are subject to change.

- We still have to decide on our budget.

- The Drone skeleton will be 3D printed to save costs (Seneca offers free 3D printing services). We may choose some eye catching colours for that.

Interface:

- We will most likely use Wifi network communication at the moment but we may expand to radio in the future. We three have knowledge on how to code packets for wifi communication but we would have to learn radio packet programming, which would take some time.

- We will most likely code the interface in C++.

- We will use OpenCV for programming the camera stream. OpenCV is an open source platform for machine vision software.

Constraints:

- Focusing on getting the Drone to fly, at the very least, and be able to talk to the Interface. Time may not be left over to add fun optional features.

- Time learning radio communication programming if we decide to switch to that

- What kind of weight the hardware parts will have (the heavier the Drone, the more expensive it is to push it to fly)

- Monetary constraints

- Overall time constraints regarding learning how to put hardware together into a Drone, programming the Drone and the interface and dealing with any problems that may happen.

- Any failed prototypes that occur that we would have to fix up.

## 

## 

## 

## 

## **(6)** **Stakeholders**

|  |  |
| --- | --- |
| **Stakeholder** | **Role** |
| Maya Filipp - Founder | Authority, Domain Expert, Technical Expert, Sponsor |
| Aaron Patterson - Admin | Domain Expert, Technical Expert, Sponsor |
| Marko Radmanovic -Admin | Domain Expert, Technical Expert, Sponsor |
| John Blair - Mentor | Onlooker, Technical Expert |
| Barbara Czegel | End User |
| User | End User |

## 

## 

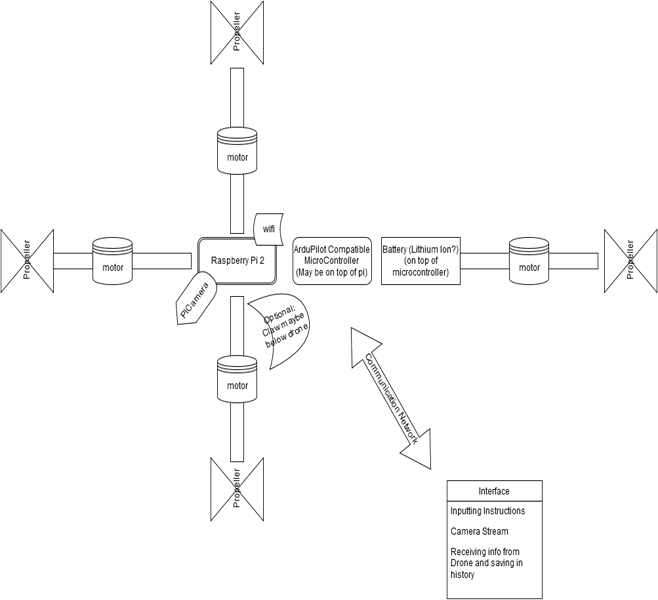
## 

## 

## 

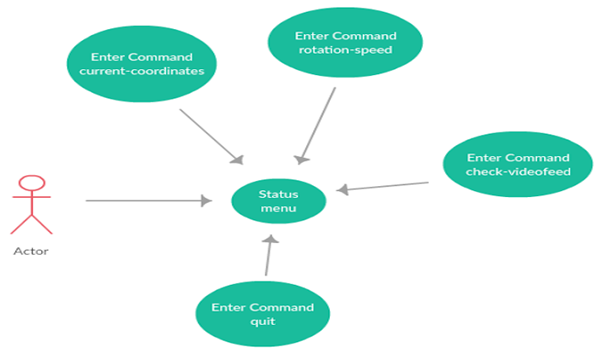
## 

## **(7)** **A Diagrammatic Representation of the System**



### **Status Menu**

The user may also view the status of the drone; for example, the coordinates, the rotation of the drone, the speed, or check the video feed. We plan to add even more to this use case as we build the drone more.



**Use Case Name:** Access Status Menu

**Actor(s)**: Drone Enthusiast

**Use Case Precondition(s)**:

· User has been authenticated.

**Use Case Successful post condition(s)**:

· User selects a menu button, redirecting them to that menu.

**Applicable Business Rules:**

· N/A

Main Flow

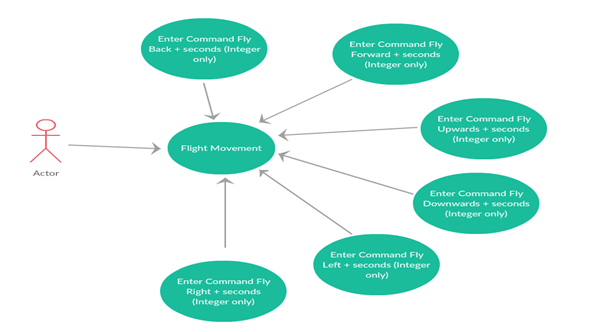
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | User enters system. | System displays numerous options for user (move object, flight movement, camera mode, view log files, land and takeoff, spinning and flipping. System also displays information about the drone’s flight, such as barometric pressure, altitude, and thrust. |
| 2 | User selects menu button. | System directs user to one of the menus selected (Move object, flight movement, camera mode, log files, landing/takeoff, spinning/flipping) |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | User quits the program. | System closes the application. |

### **Flight movement**

The user will select flight movement to make the drone move in very basic ways: up, down, forward, backwards, left, and right. The drone will move straight in the direction it’s commanded to. The user will also need to add in an integer value, which will determine the number of seconds the drone will be moving for.



**Use Case Name:** Enter fly commands

**Actor(s)**: Drone Enthusiast

**Use Case Precondition(s)**:

· User has been authenticated and is viewing a list of options on the Drone Flight movement screen.

**Use Case Successful post condition(s)**:

· User sends flight command and the drone accepts the packet and does the action.

**Applicable Business Rules:**

· Drone has to complete action before sending another command.

· Drone must be in the air flying to make a movement.

Main Flow

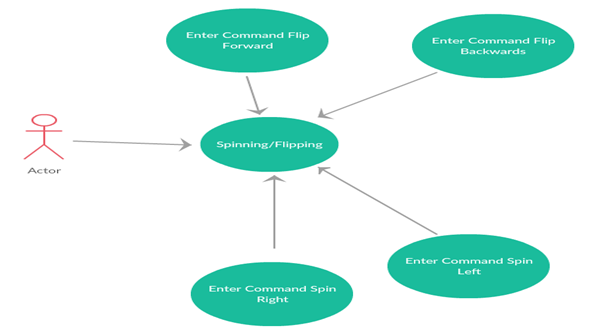
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | Requests to fly | Shows preset commands (Fly Forward, left, right, backwards, up, down and number of seconds) |
| 2 | Selects spin or flip and the direction it will perform the action in | Ensures Drone is ready to take another command  Sends packet to drone  Waits for confirmation that Drone has finished |
| 3 | Waits for Drone to finish | Confirms that Drone is finished and allows user to enter more commands. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | User enters command while the drone is grounded. | System displays error message, saying that the drone must take off to be able to fly. |
| A2 | User enters command while the drone is already executing another command | System displays error message, saying that the drone must there is already a command being executed. |

**Flipping and Spinning**

If the user selects spinning and flipping, they will be able to command the drone to perform exciting tricks. This includes flipping forward, flipping backwards, rotating a full 360 degrees to the left, and rotating a full 360 degrees to the right.



**Use Case Name:** Spinning and Flipping

**Actor(s)**: Drone Enthusiast

**Use Case Precondition(s)**:

· User has been authenticated and is viewing a list of options on the Spinning and Flipping screen.

**Use Case Successful post condition(s)**:

· User sends flight command and the drone accepts the packet and does the action.

**Applicable Business Rules:**

· Drone has to complete action before sending another command.

· Drone must be in the air flying to make a movement.

Main Flow

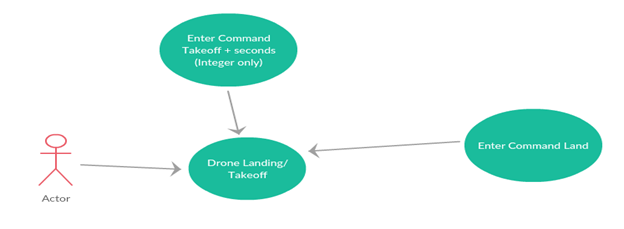
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | Requests to spin or flip. | Shows preset commands (Fly Forward, left, right, backwards, up, down and number of seconds) |
| 2 | Selects command and time it takes to execute | Ensures Drone is ready to take another command  Sends packet to drone  Waits for confirmation that Drone has finished |
| 3 | Waits for Drone to finish | Confirms that Drone is finished and allows user to enter more commands. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | User enters command while the drone is grounded. | System displays error message, saying that the drone must take off to be able to fly. |
| A2 | User enters command while the drone is already executing another command | System displays error message, saying that the drone must there is already a command being executed. |

### **Drone Landing/Takeoff**

To operate the drone, the user must choose his option to command the drone to take off in flight. The number of seconds will indicate for how long the drone will rise, similar to the command fly upwards in the flight movement use case. To land the drone, the user must also choose this option, and the drone will stabilize, and then move downwards slowly until it is on the ground.



**Use Case Name:** Takeoff/Land

**Actor(s):** Drone Enthusiast

**Use Case Precondition(s):**

**·**  Drone is either grounded or in midair.

User is at the home screen

**Use Case Successful post condition(s):**

**·** Drone successfully takes off or lands

**Applicable Business Rules:**

**·**  The drone can only take off when it is grounded

The drone can only land when it is in midair

Main Flow

|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | Requests Land/Takeoff homepage | Displays drone’s takeoff/land status and button that says take off when landed and land when taken off |
| 2 | Selects Land | Checks if the drone is flying  Sends a Land packet to drone  Drone lowers slowly until drones sensors detect a flat surface and lands  Drone sends back a completed acceptance packet  System changes status to grounded |
| 3 | Selects Takeoff | Checks if the drone is still  Sends a takeoff packet to drone  Drone turns on its propellers and flies up  Drone sends back a completed acceptance packet  System changes status to Flying |

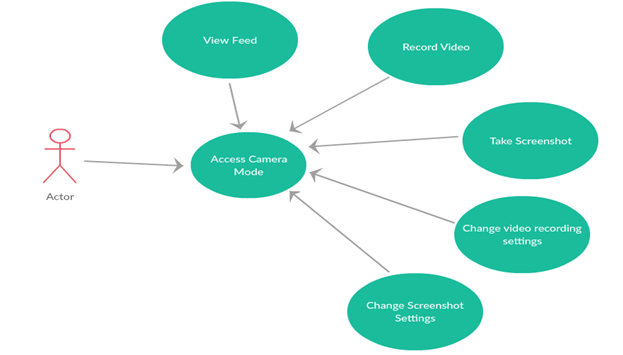
Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | Selects Land/Takeoff | System encounters error in packet, automatically invokes the landing packet |

### 

### **Drone Camera**

The drone will have a camera, and the user will interact with the camera by being able to access the camera feed from the interface menu. The user may also choose to record the video or take a screenshot, along with altering the settings between the two options.



**Use Case Name:** Access Camera Mode

**Actor(s)**: Drone Enthusiast

**Use Case Precondition(s)**:

· User has been authenticated and is viewing a list of options on the Camera Mode screen.

**Use Case Successful post condition(s)**:

· User hits the record button and records footage from the camera feed.

**Applicable Business Rules:**

· There is enough storage to store recorded video.

Main Flow

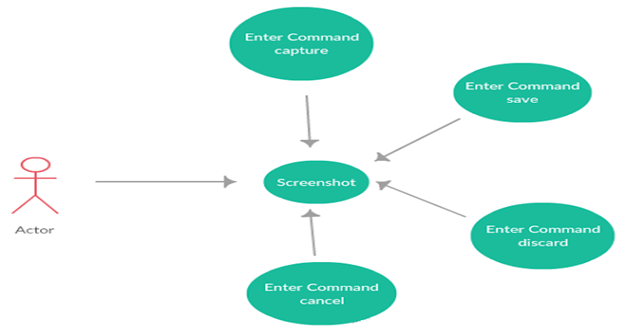
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | User requests to access camera mode. | Displays the options for the user to adjust, along with the record button. |
| 2 | User selects record button. | Ensures there is enough storage space available to hold video content. Packet is then sent to the drone and the recording begins. |
| 3 | User selects stop recording button. | The drone camera finishes recording and video is saved. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | User changes the settings and hits record. | System alters the settings objects within the packet that is sent to the drone, and the drone camera changes its settings accordingly while filming. |
| A2 | The storage space for video footage is too small. | System displays error message, saying that there must be a minimum of 50 megabytes of storage available to store videos. |
| A3 | While the video is recording, the amount of storage space for footage goes below 50 megabytes. | Camera automatically stops recording, and the system displays an error message. |
| A4 | User selects screenshot menu. | System redirects to the screenshot |

### **Capturing Screenshots**

The user may also capture screenshots using the video feed. The layout is similar to the recording video use case, with save, capture, discard and cancel being the user’s options.



**Use Case Name:** Screenshot

**Actor(s)**: Drone Enthusiast

**Use Case Precondition(s)**:

· User has been authenticated and is viewing a list of options on the Camera Mode screen.

**Use Case Successful post condition(s)**:

· User hits the record button and records footage from the camera feed.

**Applicable Business Rules:**

· There is enough storage to store recorded video.

Main Flow

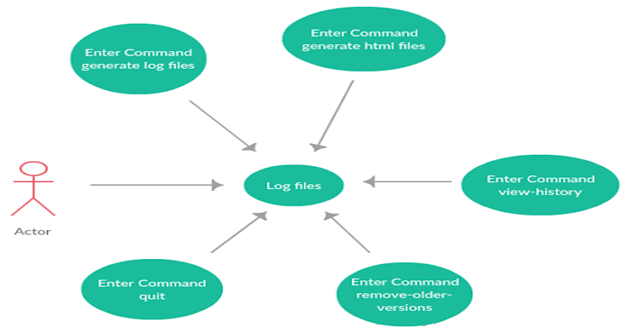
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | User requests to access Screenshot Mode. | Displays the options for the user to adjust, along with the record button. |
| 2 | User selects record button. | Ensures there is enough storage space available to hold the screenshot files. Packet is then sent to the drone and the camera records an image |
| 3 | User selects stop recording button. | The drone camera finishes recording and video is saved. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | User changes the settings and hits record. | System alters the settings objects within the packet that is sent to the drone, and the drone camera changes its settings accordingly while filming. |
| A2 | The storage space for video footage is too small. | System displays error message, saying that there must be a minimum of 50 megabytes of storage available to store videos. |
| A3 | While the video is recording, the amount of storage space for footage goes below 50 megabytes. | Camera automatically stops recording, and the system displays an error message. |
| A4 | User selects screenshot menu. | System redirects to the screenshot menu. |

### **Recording Log Files**

The user may also record log files based on the status of the drone. With these log files, the user may also look at the history of the drones actions/movements.



**Use Case Name:** Log Files

**Actor(s)**: Drone Enthusiast

**Use Case Precondition(s)**:

· User has been authenticated and is viewing a list of options on the log menu.

**Use Case Successful post condition(s)**:

· User selects one of the options within the log menu.

**Applicable Business Rules:**

· N/A

Main Flow

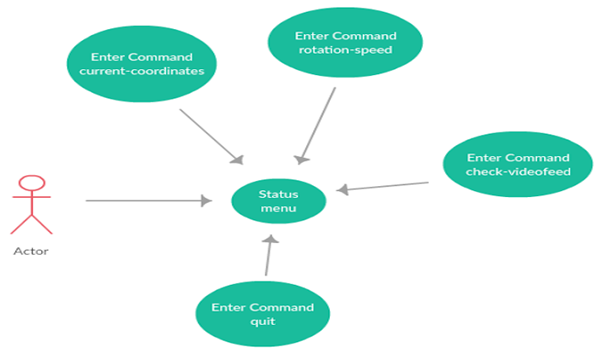
|  |  |  |
| --- | --- | --- |
|  | Actor (User) | System |
| 1 | Requests create log | Shows confirmation message when log has been successfully created |
| 2 | Selects view log | Opens the appropriate log file within the currently used application window |
| 3 | Enters quit | System confirms quit message and returns to previous mode. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | User enters cancel | System displays quit message and return to log menu |
| A2 | User enters command while another command is still working | System ignores command and continues from previously issued command |

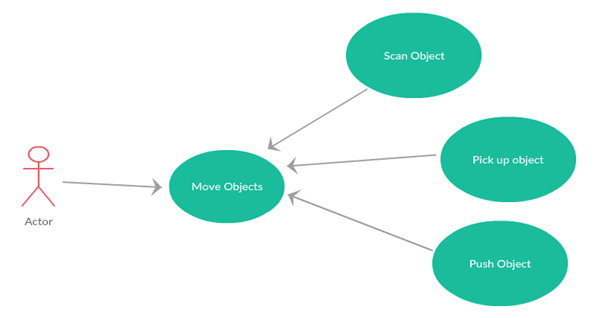
### **Status Menu**

The user may also view the status of the drone; for example, the coordinates, the rotation of the drone, the speed, or check the video feed. We plan to add even more to this use case as we build the drone more.



### **Moving objects**

Once the drone can fly competently, our next step is to build the drone to make it move objects, acting as a catering service. The scan object function will have the drone scan an object to see how large it is to see if it’s movable. If it is, the drone may pick up the object or push the object.



**Use Case Name: Moving Objects**

**Actor(s):** Drone Enthusiast

**Use Case Precondition(s):**

· User has been authenticated and is viewing a list of options on the Move object Mode.

**Use Case Successful post condition(s):**

· User is prompted to enter further commands.

**Applicable Business Rules:**

· When the drone tries to move the object it was successful, so commands can be further entered.

Main Flow

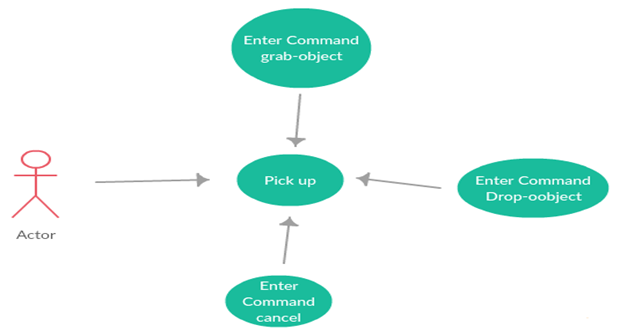
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | User requests to access Move Object Mode. | Displays the options for the user to scan, pick up, along with the push object. |
| 2 | User selects Scan button. | Scans the object and if successful allows for other options to be used. |
| 3 | User selects Pick up button. | Prompt is changed to pick up mode. |
| 4 | User selects Push button. | Prompt is changed to push object mode. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | Scan didn’t work. | Scan object was not successful user is brought back to move object mode, pick and push will be ignored. |
| A2 | User enters cancel command | System shows confirmation and returns to previous mode used. |

**Picking up objects**

In terms of moving objects, the main function the drone will focus on is picking up objects and placing them down. Once the drone has picked the object up, the user may command the drone to fly while carrying the object to move it from place to place.



**Use Case Name:** Pick up Objects

**Actor(s):** Drone Enthusiast

**Use Case Precondition(s):**

· User has been authenticated and is viewing a list of options on the Pick up Object Mode.

**Use Case Successful post condition(s):**

· N/A

**Applicable Business Rules:**

· When the drone tries to pick up the object it was successful, so commands can be further entered.

Main Flow

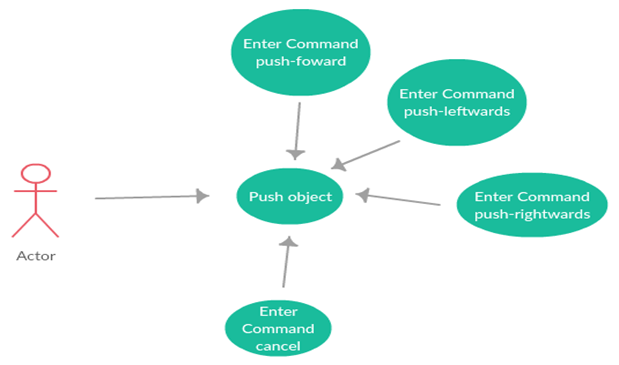
|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | User requests to access Pick up Object. | Displays the status of the attempt to pick up object. |
| 2 | User selects drop object button. | Displays success once object has safely been lowered and released. |
| 3 | User selects cancel button. | Prompt is changed to move objects mode. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | Pick up didn’t work | Pick up object wasn't successful prompt returns error and awaits further commands. |
| A2 | User enters cancel command | System shows confirmation and returns to previous mode used. |

**Pushing objects**

The user may also command the drone to push the object. The drone can push the object from different sides and angles, as outlined.



**Use Case Name:** Push Objects

**Actor(s):** Drone Enthusiast

**Use Case Precondition(s):**

· User has been authenticated and is viewing a list of options on the Push Mode.

**Use Case Successful post condition(s):**

· N/A

**Applicable Business Rules:**

· When the drone tries to push the object it was successful, so commands can be further entered.

Main Flow

|  |  |  |
| --- | --- | --- |
|  | Actor (Drone enthusiast) | System |
| 1 | User requests to access Push Object forward. | Displays the status of the attempt to push object forward object. |
| 2 | User requests to access Push Object leftwards. | Displays the status of the attempt to push object leftwards object. |
| 3 | User requests to access Push Object rightwards. | Displays the status of the attempt to push object rightwards object. |
| 4 | User selects cancel button. | Prompt is changed to move objects mode. |

Alternate Flows

|  |  |  |
| --- | --- | --- |
|  | Alternate Flow | Description |
| A1 | Push didn’t work | Push direction object wasn't successful prompt returns error and awaits further commands. |
| A2 | User enters cancel command | System shows confirmation and returns to mode prompt. |

## **(9)** **Who Did What Log**

|  |  |
| --- | --- |
| Team Member | Work completed on this Project (be specific please) |
| Maya Filipp | Concept conception, researching what to use and how to build a drone, project planning, resource gathering, editing, establishing list of products to buy, writing out coherent plan |
| Aaron Patterson | Contacting our Mentor, expanding on concept (recommending hardware), coordination with members, defining specific details/functions of the project, laying out system use cases |
| Marko Radmanovic | Researching open source platforms available for Raspberry Pi, checking compatibility in hardware, expanding on concept, defining key functions and requirements of project |

## **(10)** **The Email You Wanted Us to Include**

Our system proposal has been uploaded. We wanted to meet with you but all three of us were unavailable during this week. Could we meet with you later on if you are unable to discuss with us via email? I am attaching our system proposal as well.

We have decided to build a drone out of a Raspberry Pi computer and various parts needed for a quadcopter (such as gyroscopes, propellers, etc).

Our system is actually made up of two parts - the first being the hardware-centered drone that has onboard programming-enabled communication (most likely radio) abilities along with video feed. We may add more functionality depending on how long it takes us to build the drone. It will be of a small size.

The second part is an executable interface running on a laptop that acts as ground control for the drone. It communicates with the drone by sending instructions to it, capturing information from the drone such as video feed, replies to tests and other instructions.

Following the requirements you asked the system to have:

Client-driven: We are the clients. The idea is to essentially have a "Drone butler" that helps us with tasks when we are tired from coding. We have a mentor on board: Professor John Blair from the school of ICT, who has experience with hardware communications and programming. Luckily, he also has knowledge on drone programming as well, which will help immensely.

Non-Critical: Depending on who you ask - we're desperate for robot servants over here. Joking aside, this is purely a prototype that hopefully will be expanded on even after the three semester project planning requirement is over.

Dynamic: There will certainly be programmatical instructions sent to the drone (updating pathways, resetting, etcetera) and information sent from the drone to our interface.

Object-Oriented: Raspberry Pi, unlike an Arduino microcontroller (which we also considered), can be programmed using an object oriented programming language. Arduinos can only use C and work with few tasks while Pis can use C++/Java/etc and multi task. The ground control interface will also be coded using an OO programming language.

Persistent Storage: We will have preloaded programs saved to the Pi but most of the persistent storage will be done using the ground control interface, such as video feed saved from the drone, information replies (for example, commands from the interface like "did you find pizza" -> drone replies yes, no, or unsure) as a history of commands and responses.

Network Access: We will be using radio communication most likely, and anyone with the interface will be able to access the drone network. We might put in a password so that unauthorized requests will be disallowed.

The Interface will be an executable (although depending on how easy it is, it might be a browser based application).