|  |  |  |  |
| --- | --- | --- | --- |
| Artifact ID:  DJ-001 | Artifact Title:  Decision Justification | |  |
| Revision:  1.3 | Revision Date:  7 NOV 2019 | |
| Prepared by:  Jesse Krage | | Checked by:  Garret Gang |
| Purpose:  Describe all our different components of RPS and justify our decisions | | |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision: | Revised by: | Checked by: | Date: |
| 1.0 | Garret Gang | Jesse Krage | 18 OCT 2019 |
| 1.1 | Nick Merriman | Jesse Krage | 22 OCT 2019 |
| 1.2 | Joe Hansen | Jesse Krage | 6 NOV 2019 |
| 1.3 | Jesse Krage | Garret Gang | 7 NOV 2019 |

# References

|  |  |  |
| --- | --- | --- |
| Artifact ID: | Revision: | Title: |
| CD-001 | 1.1 | Brainstorm Notes |

# Requirement Breakdown

Control Computer Justification

*Purpose: Clearly describe each of the control systems we considered, looking at benefits vs cost. And explain why we decided to use a Raspberry Pi.*

The control computer we choose is vital to the success of this project. The key factors we had to take into account while choosing the control computer is response time, functionality, and ease of use.

The target’s current position is only updated every 500 m, an ideal control computer would start the gimbal moving towards the correct pointing vector the instant the drones’ position is updated. Due to the minimum range the target will be from the communication link positioning system, we don’t need the new position vector to be calculated instantaneously. As long as our system moves the gimbal to the proper facing within 500ms it will keep the target within the antenna’s field of view.

But due to our discussion to have the controller act as a server and be configurable thru a web-browser instead of using specialized control software; the computer needs to be fast enough to handle the load of both the gimbal pointing vector updates and the web server.

We also need to guarantee that our computer will not miss a position update because it is busy with the web server. The current positioning system we are replacing (which uses an Arduino) had a major memory leak in its initial design. While this leak has been fixed, IMSAR has a positioner out on a roof which they have to reboot once a week because upgrading their control software is more difficult than rebooting the system. Therefore, a control computer that is makes it easy to update our software/drivers remotely is a huge plus in favor of that design.

Alternatives Considered

1. Arduino

16 MHz, single core

8 kb SRAM

0 Ethernet Ports

4 USB ports

1. Banana Pi

1.5 GHz 2 core processor

1 GB ddr3 Ram

1 Ethernet Port

2 USB ports

dimensions: 92mmx60mm

1. Orange Pi

1.6 GHz 4 core processor

1 GB ddr3

1 Ethernet port

2 USB ports

dimensions: 85mmx55mm

1. Raspberry Pi B+

1.2 GHz 4 core processor.

1 GB ddr2.

1 Ethernet Port

4 USB ports.

dimensions: 85mmx56mm

# Evaluation Methods and Results

# An acceptable control computer would be fast enough to stay ahead of the drone’s position broadcast rate. It would ideally have a built in Ethernet port. It also should fit inside of the IMSAR’s weatherproof case.

Arduino:

We thought about using an Arduino because they are relatively cheap, easy to program and have really good documentation. But the downsides are that they don’t come with a built in ethernet, so we would need to use a USB to ethernet converter, or gpio-pin to ethernet. It would take a lot of extra work to make any type of Arduino work. And Arduino’s are really slow. It was the worst option by a very large margin.

Banana Pi, Orange Pi, Raspberry Pi:

Are all fast enough, have Ethernet Ports etc. In deciding between the three of these it came down to which would be easier to use. Raspberry Pi’s have a large active community of users, who are really friendly and welcoming to newcomers. Orange Pi’s and Banana Pi’s have a much smaller less active user base. Which makes it easier to find solutions /pre-built drivers/compatible hardware for Raspberry Pi’s.

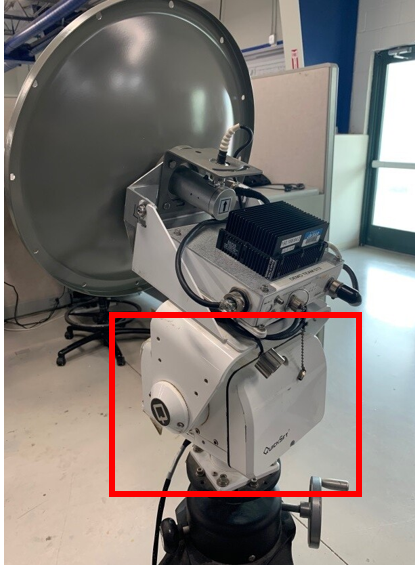
# Other Considerations

# BYU’s EE shop does not carry Orange Pi’s or Banana Pi’s. It does carry Raspberry Pi’s, which made it much faster to get a Raspberry Pi, then it would have been to get an Orange Pi.

Positioning System Justification

*Purpose: Clearly communicate the chosen concept for the positioning system and justify why the decision to use an off the shelf positioner was made.*

Concept Definition

The positioning system refers to the components necessary to physically move the antenna used to communicate with the air vehicle. It does not include the computer hardware or software necessary to control the movement. The existing radar positioning system has a tilt and pan positioner which is being obsoleted, hence the need for this Capstone project. The current system can be seen below in figure 1, with the current positioner marked with a red rectangle. This is the primary piece of hardware which will be replaced as part of this project.

**Figure 1:** IMSAR’s current radar positioning system with the tilt and pan positioner marked with a red rectangle.

# Decision Justification

In our brainstorming meeting, we came up with several concepts for the positioner. However, after conversations with IMSAR, they acknowledged they would be selecting and purchasing for us an off the shelf positioner. This will remove the necessity to develop and later manufacture our own positioner. This will allow IMSAR to feasibly produce this system, and virtually eliminate any manufacturing costs.

Using a commercially available positioner will significantly reduce the complexity of the project, as well as make it viable for IMSAR to use moving forward. By not having to design a positioner ourselves, the team will have more time to focus on the controls and making the positioner work as well as possible.

# Other Comments

As part of our concept development process alternative solutions were brainstormed. The brainstormed solutions are documented in CD-001. It was determined that using an off the shelf positioner is the most feasible option and will result in the best final product. Therefore, we will not be prototyping or pursuing other concepts related to the physical positioning system. This solution aligns with IMSAR’s desired solution.

IMSAR has communicated with us that the chosen positioner will not be available until January 2020. In order to begin prototyping and working on the control software an inexpensive pan-tilt-zoom camera has been purchased. The camera uses the same communication protocol as the final positioner that will be provided by IMSAR. This will facilitate prototyping earlier on and allow us to ultimately provide a better final product.

Server Software Justification

*Purpose: Justify our decision to host the server using NGINX and not Apache.*

As part of our system’s design, we intend to host a web page user interface on a server to allow the user to control the positioning system. This is desirable so that the UI can be accessed by any device that is connected to the same network. We decided that this constitutes a ***major decision*** because while it is important to the system architecture design, the overall design will be even if this decision is not perfect.

Methods and Results

We quickly realized that there were several viable software options for the server. Among the viable options that would run on the Raspberry Pi are: Apache, Lighttpd and NGINX. Our criteria for this decision is dictated by:

-Lightweight process that boots quickly

-Easily communicates with controller language (C++)

-Has ample documentation to help troubleshoot and optimize

Our preliminary research showed that Lighttpd and NGINX were considerably faster than Apache. While Apache is the most commonly used server software, one study reports that it is up to 2.5 times slower than either Lighttpd or NGINX. Apache becomes advantageous in high-traffic server applications where it needs to handle multiple simultaneous requests. This is not a requirement for our design and would sacrifice speed, making Apache a suboptimal for our server software.

Our research also showed that NGINX is much more widely used than Lighttpd, and as such, has more available documentation and online help. The speed and functionality of the two is comparable, but NGINX clearly fulfills the design requirement criteria more than Lighttpd.

NGINX supports the usage of a package called FastCGI, which makes for seamless communication with our controller language. We ultimately decided to use NGINX.

Server to Controller Justification

*Purpose: Justify why we connect the server to our controller using FastCGI instead of CGICC or Gunicorn*

Having our server connect to controller is essential for this project to have success. As part of our system, we need to be able to connect to our controller in order to change the direction of our gimble. For this we need to use an interface that is capable of interacting with a web server.

With this interface we can now interact with the web server we’ve configured. The server will give us information or a set of tasks we need to input. That information will ask us for our position, orientation, and target-ID. From there we want to point to the LLA(Longitude/Latitude/Altitude) of our target position. With the coordinates we can now send to the gimble.

Methods and Results

We found a few interfaces that were highly recommended: FastCGI, CGICC, and Gunicorn. We tried running CGICC, but it wouldn’t build on any of our systems, despite us following how to run it. We learned another well-made interface to build and implement was Gunicorn. However, it was in python and we felt the time to learn the language would take longer to implement another interface. Due to our programming capabilities, we wanted to find one that used C++. In our search we came to find FastCGI as the best decision.

Other Comments

Other interfaces used C++, but research showed repeatedly FastGCI is much more widely implemented.