Vehicle to Vehicle Communication for an Autonomous Vehicle

DESIGN DOCUMENT

Team Number #29
Client: Vishal Mahulkar
Adviser: Chinmay Hedge
Team Members:
Justin Wheeler
Yifan Lu
Zhize Ma
Junho Chun
Sang Uk Park
Brad Stiff
Jose Candelario
Email: sdmay18-29@iastate.edu
Website: sdmay18-29.sd.ece.iastate.edu

Revised: 11/27/17-V2

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

As part of our senior design we have to create the means for the communication of two vehicles. For this we have come up with several solutions throughout our design and testing process. We have looked into DSRC, Cellular Communication, Raspberry Pi's over a LAN network, and Xbee transceivers. All these are viable options and several can complement the others pretty well. In the following we will talk about how we have tested some of these and which one we will use. We will also discuss some other aspects of the project that we have taken on.

1.1 Acknowledgement

First off we would like to formally thank Dr. Hegde for guidance on our project and the insightful advice on how to approach different obstacles of our progress. Also we would like to thank Vishal as our project manager who made sure we stuck to our milestones and meeting with us every week to make sure we were working towards our goals. Both of the faculty members have been a significant presence to our progress in this group project.

1.2 Problem and Project Statement

General Problem:

Our project revolves around transportation, but a transportation system that ranges more than simply accommodating people from place to place. When we think about transportation of any kind, the first thought that comes to our minds is being inside a certain vehicle, whether it be a person or an object being transported. We would normally think that a person would be manning that certain vehicle: however, our approach is to transport both a person and a vehicle without having a person operating each one.

Having one person operate multiple vehicles does offer a new aspect to transportation. It's similar to having a locomotive pulling multiple cargo carts except this one isn't on tracks, it's on the open road. This is very important because for years we have relied on trains for mass transportation of goods, but they are limited to railroad stations. With the rapid growth of local roads and highway systems, we also rely on big trucks but their cargo is much more limited compared to trains and this is where we offer a solution to mass local shipping. Having a lead truck driver lead multiple autonomous following trucks, we are not limiting mass transportation on the tracks anymore, it can be on on any road or highway which saves both time and money for shipping companies.

General Solution:

Replacing big trucks drivers is only one of the main uses for our project. Our goal for this project is to have an autonomous car capable of following a manned lead car. Our autonomous car functions mainly off of information gathered by multiple sensors such as location of the lead car and the obstacles that may lie in between the two cars. This kind of project requires several teams such as a mechanical team, robotics team, controls team, and an electrical team.

As part of the electrical team our role is more defined. We have two main tasks to help create this autonomous vehicle solution. The first task and most important is sending the location of the lead vehicle to the following vehicles. We currently have three different methods that we would like to implement to achieve this. First we plan on using two transceivers attached to the gps on the lead vehicle as well as one attached to the ROS on the following vehicle. We originally planned on using DSRC but have decided to go two XBee transceivers. We also plan on using 4G LTE cellular to transmit the location for when the vehicles are at farther apart locations. We also are using a raspberry pi which is similar to a small computer. This should allow us to retrieve the location information directly from the GPS. We have currently used an XBee and a the raspberry pi to transmit the information. We are currently working on interpreting the data coming out of the GPS and creating something useful out of it.

We also are doing any hardware and software small projects that may be needed of us. Throughout the project we have been soldering on wires to the right connectors. Finding parts for the other groups or anything that they need that we may have. One example is that we attached the proper connectors and power source to the radar that we are using. At the end of this project we will have created a reliable way of sending messages from one vehicle to the other. We will also have a fully autonomous car.

1.3 operational Environment

For our simulation and testing, we plan on doing the prototype testing process in an empty road. Since our data transmission is strictly limited between the two vehicles, our first evaluation would preferably be on a open field where we can test the trajectory planning of the following vehicle with the lead car going one direction. We would be most interested in the Xbee sending the information from the GPS at a distance of at most 300 feet. We will do this testing first by testing that information can be received at 50ft, 100ft, etc. Once we are able to reach 300ft we will consider the communication complete and ready for improvement. We expect the vehicles to eventually work in any type of environment which means that we need to cover all our components to all types of weather. The mechanical team will handle all the insulation from the weather.

The lidar sensor and radar which will be important for measuring the distance between the lead car along with other obstacles will be resistant to rain and snowy condition in which we will have both our cars in. It is important for us to know that the conditions in which our cars will operate do not give us incorrect data in the transmission process. Fortunately, all of the main hardware that will be responsible for transmitting data to the ROS and controls will be in the interior of the car where harsh weather conditions won’t disable any intended functions.

1.4 Intended Users and uses

We can look at two different uses for our project: the larger project an autonomous car following the coordinates of a lead vehicle or destination, and our more specific project which is creating the communication between the lead vehicle and the following vehicle.

For the larger project we see the end user being a company with many cargo trucks that wish to cut down on number of drivers. We can also see the army or other armed forces that would like to transport a whole base or move supplies, having an autonomous car that only requires one driver may be beneficial. The uses are mostly just moving materials and goods with less expense. Someone with a lot of money may as well wish to move their car collection all together at once such as in a parade.

For the more specific project that we are dealing with the user can vary much more. We can see anyone dealing with communication wanting to buy our final product for their own communication. There are many road object avoidance applications that may benefit from our product. Road safety applications as well as any type of medium range communication may wish to use our XBee communication method meanwhile longer communication will benefit from our 4G LTE application.

1.5 Assumptions and Limitations

Assumptions:

For our end product, we are assuming that the main power source for all of our electronic hardware sensors will be a 12V battery. The reason that the power source will have to be 12 volts is due to the fact that the radar has the maximum voltage intake of 24 volts but that we have created a 12 to 24 volt converter for it. The rest of the sensors that operates along with the radar is below 24 volts and in order to avoid over-currents, we may be using multiple DC to DC converters to lower the voltage intake from the primary battery Or step ups depending on which sensor we are powering. All the components vary from about 5 volts to 24 volts.

Our second assumption is that the following car must be able to receive the data from the lead car from an average distance of 200 feet. This distance is not set to stone, but our primary goal is to have a data transmit range that is long enough so that incase the following car is stopped by an obstacle/traffic light that is between the following and the lead car, the lead car can transmit its X,Y coordinate location so that the following car can get a rough approximation of where the lead car is heading.

Our last, but not least assumption will be that the communication between the ROS and sensor receivers will be operated using two computers that one has ROS installed on it and the other has python on a raspberry pi. For the prototyping, this is very important because we need to have one person monitoring what data is being transmitted while the other person monitors if the data transmitted is consistent with the data received. This way we can see where the data errors are coming from if we were to encounter any. This brings us to our limitations regarding the transmission of data. We plan on installing a small monitor on the raspberry pi both for debugging and for programming as well as any other uses that may come up.

LIMITATIONS:

Our biggest limitation currently is the range limit of the data transmission. Initially, we were planning on using a DSRC transmitter which is originally used as a communication transmitter for vehicle to vehicle communication. The DSRC modules emit a frequency of 5.9Ghz which enables a range of far over 200 feet.[2][3] Unfortunately these V2V communicators were extremely expensive and also government regulated, which forced us to think of another way to build a transmitter with possibly a smaller frequency. Currently we are using the XBee transceiver which has a decent range of 100 meters to 300 meters. This is limiting but for the purpose of our project should suffice.

1.6 Expected End Product and Deliverables

The end product will have a raspberry pi attached to an XBee in order to transmit the information to the following vehicle. It will have a fully working screen and keyboard attached in order to make changes to the amount of data that the GPS actually sends out. It will also be attached to a power source attached to the actual vehicle. We will attach another XBee to the following computer as well.

We will also have a data sheet with all the code we used to set the GPS to the right modes so that the user may know what is going on. We will create a user friendly interface to make it easier on most cases. We will also have information on how everything is attached to each other.

We hope to later on include a 4G-LTE method of transmitting the information as an optimization for when the vehicle gets too far away from the other vehicle. When we implement this we will also need to include the code required for the app we will create for it.

We will also do videos of our working final project working out in a parking lot. We will test it at different ranges to see show how far apart the vehicles can actually be. Finally we are actually going to be implementing this device on two golf cars that the other teams are currently also helping automize.

2. Specifications and Analysis

Since our main role in this project is data transmission, we have been thinking of ways to have a smooth data transmission process between the GPS to the ROS system on the following vehicle. Below, you will see one of our methods that we wanted to implement to achieve a max range. Initially, we did not know that a DSRC system would be so expensive and then had to change to a different transmission system with lower frequency. Nevertheless, our group plans on following this outline below in order to achieve a smooth method of data transfer for the car locations. We have gotten two transceivers called XBees which we have been able to use to transmit over at least 120 meters.

So far we have been able to connect the whole system of transmission together and are able to transmit the GPS data to a remote computer roughly 120 meters away. We have tested out all components such as the XBee, raspberry pi, and the code necessary to receive the data. At this point since all that works we are working on adding some optimizations to the system. We tested out the XBee and raspberry pi first by using a Matlab program as a simulation for the GPS. After testing we concluded that we should be able to transmit data through these devices but we wanted to make sure they could handle the information being sent by the GPS so we tested it on the actual GPS. We are currently analysing the data that we got but it looks very much like it is what we expected.

We are analysing this data that we received and comparing it to the datasheet. We are able to see the patterns that we expected and some new ones that we are unsure of what they mean. The next task in hand is to figure out how to convert the hexadecimal data that we are receiving into useful information that the controls and robotics team may be able to use.

2.1 Proposed Design

Our team proposed several possible solutions to solve the problem of communicating GPS data from the lead vehicle to the following vehicle. Each solution has both pros and cons which need to be looked at. We have come up with a list of these to determine which solution out of the three is the best.

The first includes the use of two microcontrollers. One will be attached directly to the GPS on the lead vehicle, while the other would be directly connected to the NVidia PX2 on the following vehicle. The purpose of the microcontroller on the lead vehicle would be to take the serial data, compress it, and send it to the DSRC transmitter. The signals would be transmitted to a DSRC receiver which would be hooked up to the second microcontroller on the following vehicle. This microcontroller would convert the signals into data which could be read from ROS running on the Nvidia PX2.

The second solution is to use a Raspberry Pi on the lead vehicle which could connect directly to the GPS. The Raspberry Pi is capable of running ROS on it which would allow it to be a network node. This would make it very easy to communicate the data back to the NVidia PX2 running ROS on the following vehicle. The only downside is the range at which these two devices would communicate through a local network wouldn't be as big. Another option is to attach transceivers such as XBees unto the raspberry pi and use python coding to upload information from the GPS and transmit it to the following vehicle.

The third solution would be to use already built transmitters and receivers called Xbees. These would connect directly to the GPS on the lead vehicle and to the NVidia PX2 on the following vehicle. The connection using this method would most likely be the simplest since the data coming directly from the GPS would be sent directly to the NVidia PX2. For this reason, it would be a lot easier to debug if things are going wrong. This could save us a bunch of time in the long run.

So far, our analysis has concluded the first solution to be the hardest to debug. If something goes wrong, it would be hard to track which section in the communication wasn't correct. It would also have the best range out of the three. The second solution would suffer due to the range factor. If the lead vehicle gets too far ahead, the connection would be lost and the following vehicle wouldn't know where to go. The last option was found to have several problems with actually receiving information from the GPS. It gave no option to power the GPS. We then had to do the second part of solution two where we combined the XBee and the raspberry pi. The code that we added to the raspberry pi was efficient luckily to the fact that the GPS sends information at roughly 10 Hz. Our code was fast enough to handle this type of information coming in.

2.2 Design Analysis

So far we have completed the ability to transmit the GPS data to another computer 100 meters away. We first tried doing this through using an arduino and the XBees that we have. We had troubles powering all devices involved using the arduino so we ended up switching to using a raspberry pi. With this new method we have been able to communicate the information from within the GPS. We are currently transmitting only raw data so the next step is to package that data into something that may be more useful for the controls and robotics team. Another next step is that since we are now using the raspberry pi we can actually attach a small monitor with keyboard and be able to program the GPS from there if we wish to change the settings that it has. This change with a good user interface should make a great addition for testing and later uses.

So far our strengths for using this design is that we will be able to more easily change the settings of the raspberry pi as well as that of the GPS without having to find a lab or using a personal computer. This allows on site debugging. Some weaknesses we believe to be that we need to find space in the vehicle for the monitor that we may wish to implement which was not taken into account earlier by the mechanical team since it's relatively new. We believe that the proposed solution will be very versatile when implemented.

3 Testing and Implementation

For our project we have had to make several tests and will do several more for other parts of the project. The tests include equipment capabilities tests, connections tests, and software tests. We have tested the XBees capabilities of transmitting such as distance and type of data being sent. We also have tested arduinos and raspberry pi. The testing of these two have consisted of testing the code that we write and testing the capability of them interacting with the GPS system. We have also tested and implemented the powering and usefulness of the radar. We are currently working on powering all the components on the vehicle as well and a small group of ours is currently gathering data for that.

XBee:

First when we got the XBees we wanted to verify the distances that the XBees could transmit information compared to that in the data sheets. The datasheets said that it could transmit up to 300 meters in open range and 100 meters when there are buildings in between. In order to test this we plugged two XBees unto two different computers. We then walked roughly up to 120 meters while communicating with the XBees. throughout this process we were still able to receive information. The library building was between the two XBees at the longest distance and at roughly 120 meters is when information started to get cut.

We also tested the type of information that we could send through the XBee. We first tested simple text messages. We also tested HEX values since that is the type of messages that we are going to be sending. We tested the HEX by creating an arduino code that only sent messages in HEX and received the information on the other side. We also checked if we could receive in packages. It is able to do so but it seems more feasible and easy to check if there are any errors in the data sent if we just send HEX data by itself and not in packages.

Arduino/Raspberry Pi:

We created a simple serial-in code in arduino that we hoped would allow us to communicate with the GPS. The code consisted of bringing in one byte at a time from the GPS and then sending it as HEX value through the XBee. We first tested this in the lab by hooking up the arduino to the computers in the lab. These computers were running a Matlab code that simulated the GPS sending out serial bytes. When tested in the lab it was straight forward but once we went to use the actual GPS we had troubles. We believe that this had to do with the way we were powering the devices. The GPS can only be powered through the same port that transmits the data so cut up a couple USB cables to try and power everything but since there were three devices trying to be powered through these floppy cables every time we tried it on the GPS we would fail. At this point we decided to change from an arduino to a raspberry pi. The reason for doing this is that the raspberry pi has several USB inputs so we don’t have to create special wiring for the GPS and for the XBee. It also has a higher tolerance for current but can work at a lower current level. This higher current level allows for more devices to be plugged into the same device and still have sufficient power to work. We created a Python language code that took in information in bytes from the GPS. The python code is very similar to the code that we had originally created for the arduino in function. The only complications with the raspberry pi solution is that we need to get an interface for it in order to be able to control when it starts implementing the python code. We are looking into interfaces and keyboards and other alternatives.

Radar:

For the radar we first needed to connect the proper cables to a DB-9 connector in order to read the information that it needed. We also had to implement a 12-24 DC converter. This is because the radar (ESR) only works with 24 volts but in the vehicle it will only have access to a 12 volt battery. Once we had it working we then tested it by taking data of walls and objects in front of walls to see if we could tell them apart. The controls and robotics team are going to use this data to understand how to tell if there are objects in front of the vehicle or not.

Powering:

We are powering all the devices in the following vehicle using a 12 volt battery source. There is a total of four 12 volt outputs available and we are currently gathering information on the requirements of all the components. Once we know all the requirements we will start connecting the devices together. We plan on simulating it in the lab first to see if we need to add extra resistors to bring down the current or vice versa. Once we get a circuit that we believe will meet all the components requirements we will start connecting and test out the larger circuit. We will add circuit breakers so that we don't fry any of the more expensive equipment that we have.

We first had in mind using the arduino as the receiving and transmitting component on the GPS side but once we discovered through testing we had to replace it with the raspberry pi. We retested then and were satisfied with the results. We plan on adding an interface to it to make it simpler to use.

3.1 Interface Specifications

The power supply for sensors such as radar, lidar, camera is need for 24v. For GPS, it has input voltage range between 4.5-34v.

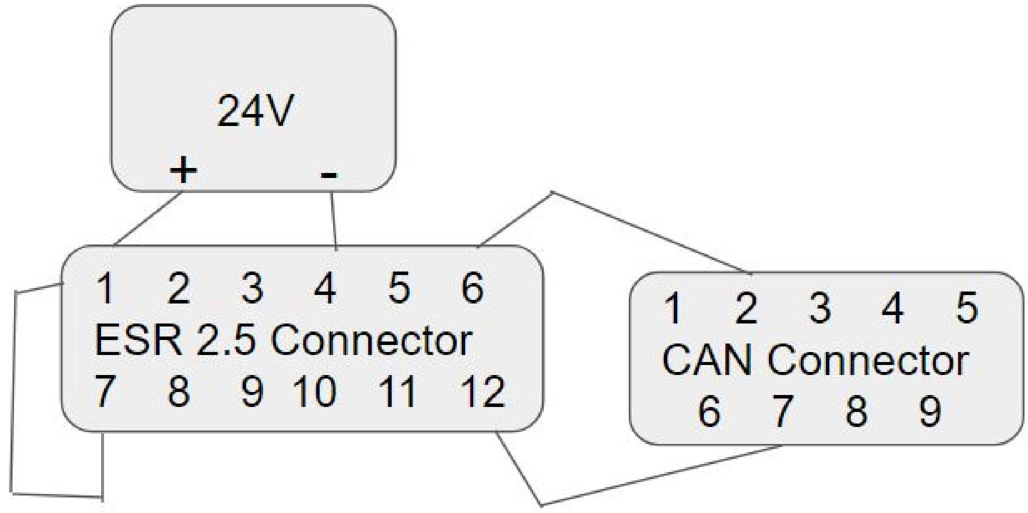
• Camera Interface Specification

Interface: USB 3.0

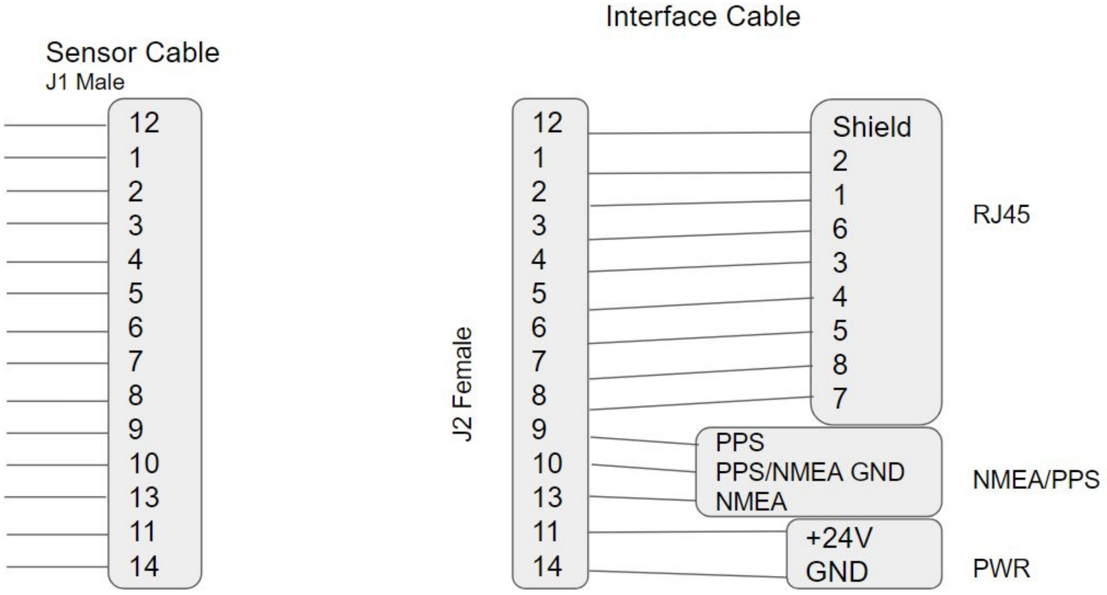
Power Requirement: 8-24V via GPIO or 5V via USB 3.0 interface

Pin Number	Singal	Port Color
1	Battery (+24V)	Red
2	USB D+ (Green Wire)	Green (USB)
3	USB D- (White Wire)	White (USB)
4	Ground	Black
5	USB Ground (Black Wire)	Black (USB)
6	PRVCANL	Green
7	Ignition (+24V)	White

8	**USB +5V (Red Wire)**	**Red (USB)**
9	**VEHCAN L**	**Blue**
10	**VEHCAN H**	**Brown**
11	**VEHCAN Shield**	▨
12	**PRVCANH**	**Orange**



<u>The Lidar</u>



1	2	3	4	5	6	7	8	9	10	11	12	13	14
Orange	White	Green	Yello w	Blue	Clea r	Brown	Light Blue	Violet	Tan or Pink	Red	Bare	Gray	Black

Power supply	24VDC/1.5A(max)
Protection	Short Circuit/Over Current/Over Voltage
Ripple	1-2%Vpp

We plan on adding a screen as well as a keyboard to the raspberry pi in order to interact with the GPS on a more versatile level. All the connections to the raspberry pi are straightforward USB connections both to the XBee and to the GPS. We will see what connections we get once we decide on a screen interface.

3.2 Hardware and software

Hardware:

GPS: It will be used to let the following vehicle know where to go. We will send the latitude and longitude location of the lead vehicle so that the following vehicle has a location to where it must travel.

Radar,lidar and camera: Help detect the surrounding environment near the following car. They will be used for object detection and avoidance. Having multiple options for detection will allow each of them to complement each other in different types of road conditions. Some work better in fog and different temperatures than the others do.

Xbee: The XBee is a transceiver that can work in ranges from 100 meters to 300 meters.[1] This transceivers are useful for transmitting the information over the distances that we require for the communication.

Raspberry Pi: The pi is a small computer that allows us to access the information coming out of the GPS as well as be able to program the GPS itself. We are using Python on it.

General Lab Equipment: Power supplies and multimeters have allowed us to test connections for the sensors when we have had to assemble a different type of connections on them.

Software:

ROS: It is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. It used to transmit the data language for needed information.

Python: We are using python programming language for most of the coding that we did on the raspberry pi as well as on ROS. Python is easy to use and understand and allows us to do what we need to.

Matlab: We have used matlab to simulate the GPS in the lab. It is not exactly a GPS but allows us to closely simulate it for the sake of testing our other components.

3.3 Process

In order for us to fully test how each of the components around the following car gathered and output data, we had to first thoroughly test each hardware and see how each of the outputs were formatted. Then we had to put them together as explained in part 2 and test the connections and ability of the components to talk to each other. Throughout multiple testing attempts, we found certain hardware obstacles other than limitations such as serial protocol which prevented us from receiving the correct serial data. Overall, our final solution came down to using a general component that shared the same serial protocols with both the GPS and the XBee. This solution was using a raspberry pi which had the same serial protocol ports (USB) for the GPS and the XBee.

GPS: The GPS called "MTi-700" used a computer program called Xsens to output its data. The output program would have multiple options of output such as altitude, speed, velocity and position. The Xsens manager would also output a graph along with the raw data with a baud rate of

115200. The example of the raw data that we collected along with the different modes is shown below:

(All configurations enabled)

```
{(PacketCounter, 2 bytes, 44551), (SampleTimeFine, 4 bytes,    4508277),
(Quaternion|Double|ENU, 32 bytes, (q0:   0.99978242, q1:  -0.00530635, q2:
0.01911533, q3:  -0.00644657)), (DeltaV|Float, 12 bytes, (x:  -0.00085376, y:
-0.00024180, z:   0.02469990)), (Acceleration|Double, 24 bytes, (accX:  -0.34148182,
accY:  -0.09670638, accZ:   9.87995956)), (FreeAcceleration|Float, 12 bytes, (freeAccX:
0.03472491, freeAccY:   0.00987460, freeAccZ:   0.08325957)), (AltitudeMsl|Double, 8
bytes, altMsl: 300.74200000), (AltitudeEllipsoid|Double, 8 bytes, altEllipsoid:
270.77932285), (LatLon|Double, 16 bytes, (lat:  41.99774220, lon: -93.63300438)),
(RateOfTurn|Double, 24 bytes, (gyrX:   0.00105351, gyrY:  -0.00188649, gyrZ:
0.00249594)), (DeltaQ|Float, 16 bytes, (q0:   1.00000012, q1:   0.00000132, q2:
-0.00000236, q3:   0.00000312)), (VelocityXYZ|Double|ENU, 24 bytes, (velX:
0.17639036, velY:  -0.00129774, velZ:   0.01468735)), (StatusWord, 4 bytes,
00000001100000000000000001000111)}
```

(Just GPS enabled Floating point 32 bit)

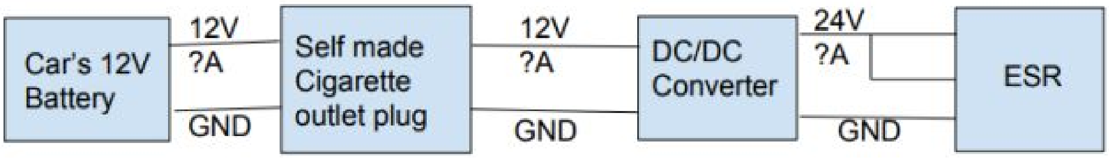
```
{(PacketCounter, 2 bytes,  9402), (SampleTimeFine, 4 bytes,    5266523), (LatLon|Float,
8 bytes, (lat:  41.99771881, lon: -93.63301849))}
```

(Just GPS enabled fixed point 12.2 bit)

```
{(PacketCounter, 2 bytes, 63411), (SampleTimeFine, 4 bytes,    6620371),
(LatLon|Fp1220, 8 bytes, (lat:  41.99771595, lon: -93.63331032))}
```

The data that we collected through the Xsens manager shows us the length of each data output for different measurements. By gathering this data, it gave us the idea that this program converts the raw serial data into the ones shown above. This part is useful for the following vehicle knowing where it is at. We then decided that for transmitting the data from the lead vehicle it might be a better idea to just transmit the raw hexadecimal data. We first tried using an arduino to receive this information but ended up needing to change to a raspberry pi due to input options being better suited to the raspberry pi.

Radar/Lidar: For these components, our group is mainly responsible for powering them while the controls team is responsible for using the data for trajectory. For the powering of the radar, we utilized a cigarette output plug and a DC to DC converter to power the ESR radar system. The diagram that explains the process is shown below:



For the Lidar, we were not able to get to powering it, but since it also uses a 24V DC power similar to the ESR radar, we plan on using a similar method and extending the power applied to the ESR to also power the Lidar. We have tested getting information from it and feeding it through ROS. We are able to get the information and manipulate it now for the controls tea.

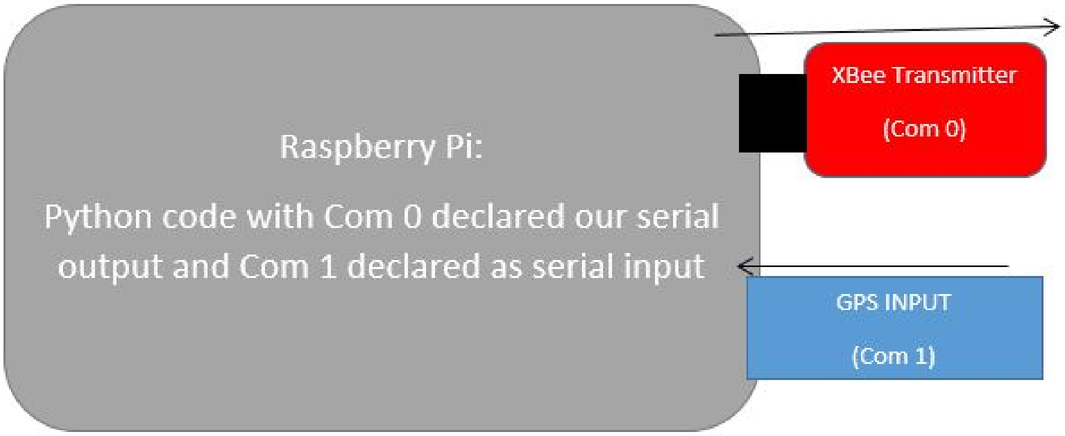
XBEE: We ran multiple tests with the Xbee in which we connected the arduino to transmit the raw GPS data between the coordinator and end Xbees. We used the arduino between the XBees because the GPS outputs raw serial data in 16-bits while the Xbee can only receive a limit of 8-bits. In order for the Xbee to receive the raw data in packages, we programmed the arduino to receive the serial data and be able to package the 16-bits. The serial data followed a rule in the Xsens manager in which there was a start and end of each serial that was transmitted by the GPS.

```
11-09-2017 19:36:47.271,-,API," ,0013A2004163F0FF,XBEE PRO
802.15.4,10EF,COM4 - 115200/8/N/1/N,1"

11-09-2017
19:36:47.280,45691,RECV,00698100002400353564323535643235356432353564
32353564323535643235356432353564323535643235356432353564323535643235
356432353564323535643235356432353564323535643235356432353564323535 64
32353564323535643235356432353564325A
11-09-2017
19:36:47.308,45692,RECV,002F8100002400353564323535643235356432353564
3235356432353564323535643235356432353564323535F0
11-09-2017
19:36:47.308,45693,RECV,00698100002400643235356432353564323535643235
35643235356432353564323535643235356432353564323535643235356432353564
32353564323535643235356432353564323535643235356432353564323535643235
3564323535643235356432353564323535 5A
11-09-2017
19:36:47.308,45694,RECV,00698100002400643235356432353564323535643235
35643235356432353564323535643235356432353564323535643235356432353564
32353564323535643235356432353564323535643235356432353564323535643235
35643235356432353564323535643235355A
11-09-2017
```

We also performed tests on the XBee to check the range at which it would work. These tests gave us confidence that the operating range would be within the required amount of 120 meters. We tested several types of information at different speeds leading up to sending the GPS data as the final test to see if it was able to handle that type of data.

Raspberry Pi: The raspberry pi was used for the final attempt for data transmission. Previously with the arduino, we ran into serial compatibility problems, where the arduino and the GPS were not able to have correct data being transmitted to it. The issue was that the USB and arduino used different standard/protocols in which each device has a different serial port. To address this problem, we used a Raspberry Pi, which was a programmable mini computer which has the same USB serial port as the GPS. We programmed the raspberry pi to be able to receive serial data from the port in which the GPS was plugged in and then use the second USB port to connect the Xbee in which was used to transmit the GPS serial data to the end Xbee.[7] Using this method, we finally got the correct serial data with the start and the end (Low Level GPS Communication Format). Our final output will be included in the results.


Raspberry Pi:
Python code with Com 0 declared our serial output and Com 1 declared as serial input
XBee Transmitter
(Com 0)
GPS INPUT
(Com 1)

3.4 Results

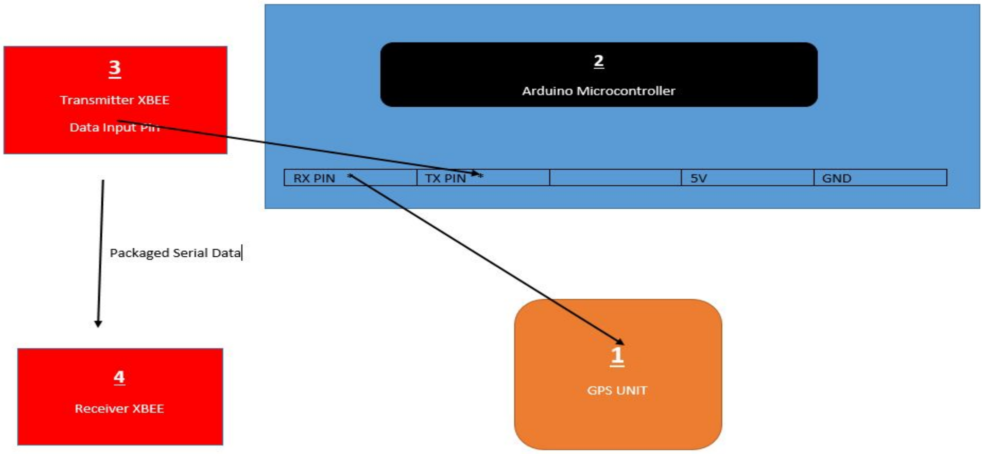
So far, we have been able to transmit and receive the correct data format from the GPS through the XBee. Utilizing the raspberry pi to bridge the different protocols between the GPS and the XBee, we were able to get the desired serial data wirelessly through the XBees. The low level protocol indicates that the GPS's serial data follows this format[9]:

PREAMBLE	BID	MID	LEN	DATA	CHECKSUM

The above is how we expected data to be received from the GPS. Once we received the serial data the first four characters should 0xFAFF as part of the Preamble and Bid. Then Mid gave us the message identifier followed by the len that would give us the length of the message. The Data would be given in relation to length. Which in this case would be the raw serial code. Checksum would then add the total bytes in the previous five sections and give us the sum.

We have also implemented some python code in the receiving computer that allows us to convert the low level bytes to something more readable like the actual location in latitude and longitude.

Our initial tests required an arduino and the XBee, with the Arduino receiving data from the GPS and then sending the packaged data out to the transmitter XBee. The diagram below shows how we initially tested:



Overall, we were able to get data, but the data that we received was inconsistent to what the low level communication format that was shown above. Through multiple tests and various changes in our original arduino code we were unable to get the desired data format which we later realized was due to the different data protocols in which the GPS and arduino had. Both the GPS and the Arduino had USB protocols; however, opening the usb ports of the GPS and connecting with pins caused it to have erroneous data since they were not the similar protocol. The data that we received using this method is shown below:

```
11-09-2017
19:36:47.280,45691,RECV,00698100002400353564323535643235356432353564
32353564323535643235356432353564323535643235356432353564323535643235
35643235356432353564323535643235356432353564323535643235356432353564
32353564323535643235356432353564325A
19:36:47.308,45694,RECV,00698100002400643235356432353564323535643235
35643235356432353564323535643235356432353564323535643235356432353564
32353564323535643235356432353564323535643235356432353564323535643235
35643235356432353564323535643235355A
11-09-2017
```

From these results, it was clear it was not the data format we were looking for since the start bit does not display “oxFA” and rather number values. Overall, it took us a while for us to realize that this issue originated from having inconsistent device protocols. Since this was the first time implementing fully assembled, functional hardware to a microcontroller, there were many specifications that we didn't take into regard which lead to these issues. From multiple attempts of testing and modifying the arduino code, we learned it was a hardware issue rather than software. Since both the GPS and the XBee used a serial protocol (USB) we needed something that could bridge the two protocols into a single processing system which could be programmed to receive and transmit serial data.

As stated before, we resolved this issue by using a raspberry pi, which we used python to program the pi to receive the serial data from the gps connected port and then transmit that to the XBee connected port. We would use the receiving XBee and connect it to a laptop to observe the raw serial data that we received. The data that we collected is shown below:

0xfa	0xfa	0xfa	0xfa	0xfa	0xfa	0xfa	0xfa	0xfa	0xfa
0xff	0xff	0xff	0xff	0xff	0xff	0xff	0xff	0xff	0xff
0x36	0x36	0x36	0x36	0x36	0x36	0x36	0x36	0x36	0x36
0x2a	0x2a	0x2a	0x2a	0x2a	0x2a	0x2a	0x2a	0x2a	0x2a
0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10
0x20	0x20	0x20	0x20	0x20	0x20	0x20	0x20	0x20	0x20
0x2	0x2	0x2	0x2	0x2	0x2	0x2	0x2	0x2	0x2

Overall, we were very relieved to finally have received the correct serial data format described in the Low Level Communication diagram. The first bit started with "oxfa" which indicated the start along with the data length. With this method, we were finally able to remotely receive and transmit the data at a reliable baud rate.

4 Closing Material

4.1 Conclusion

In conclusion, our goal is to establish safe communication between both vehicles. To accomplish this, we came up with four possible avenues. First, was to use DSRC for communication between vehicles. This option provided more precision than others, but it came at the cost of high data conversions. Our next option was to use cellular phones, this provided more than enough distance, but the data would be transferred too slow due to buffering between satellites. This option also gives us a network outside of our control, and can not manage any data after its packaged to be sent. Another approach, was to use the Raspberry Pi on both vehicles for transmissions. This really only added a step, due to all the attachments we needed for communication had a stand alone option as well. Lastly, The Digi Xbee Pro, offered the distance needed along with the ability to plug directly into the Nvidia PX2.

As of right now, our best plan of action is to use the Xbee along with the raspberry pi for our communications between the lead and follow vehicle. The XBee was chosen, because it met the distance requirements from our client, and it allowed for the most practical implementation. After much testing with the arduino and raspberry pi we have finally gotten a working option with the raspberry pi. We currently are able to transmit the whole location of the GPS and are looking into optimizing it at this moment.

4.2 References

Digi Xbee S1 802.15.4 RF Modules [1]

https://www.digi.com/pdf/ds_xbeemultipointmodules.pdf

Car and Driver - Vehicle-to-Vehicle Communications Are the Next Big Thing in Auto Safety[2]
https://blog.caranddriver.com/vehicle-to-vehicle-communications-are-the-next-big-thing-in-auto-safety/

• V2V DSRC communication systems

Automotive News - V2V finally on its way, but is it too late [3]
http://www.autonews.com/article/20161226/OEM06/312269996/v2v-finally-on-its-way-but-is-it-too
late%3F

• 5G V2V long distance communication

Toshiba - RF ICs for Vehicle Communication Service [4]
https://toshiba.semicon-storage.com/eu/product/automotive/dsrc-ic.html

• RF IC communications diagram

Toshiba - Global Sales [5]

https://toshiba.semicon-storage.com/eu/corporate/about/global-sales.html

DigiKey - Radio Links for Driverless Cars [6]
https://www.digikey.com/en/articles/techzone/2016/dec/radio-links-for-driverless-cars

• V2I vehicle to infrastructure communication

Lora/GPS Hat for the Raspberry Pi [7]

http://wiki.dragino.com/index.php?title=Lora/GPS_HAT

NVidia Drive PX [8]

https://iowastate.sharepoint.com/sites/followme/...

MT Low-Level Communication Protocol Documentation [9]
https://iowastate.sharepoint.com/sites/followme/...

MT Manager User Manual [10]

https://iowastate.sharepoint.com/sites/followme/...

MTi 710 User Manual [11]

https://iowastate.sharepoint.com/sites/followme/...

MTi User Manual [12]

https://iowastate.sharepoint.com/sites/followme/...

Xsens - MTi 100-series Data Sheet [13]

https://iowastate.sharepoint.com/sites/followme/...

Project Requirements from Client [14]

https://iowastate.sharepoint.com/sites/followme/...

4.3 Appendices

The actual GPS model we are getting information from:
https://www.xsens.com/wp-content/uploads/2013/11/MTiG_User_Manual_and_Technical_Documentation.pdf

Attaching multiple machines on ROS tutorials:
http://wiki.ros.org/ROSberryPi/Setting%20up%20ROS%20on%20RaspberryPi

http://wiki.ros.org/Distributions

http://wiki.ros.org/ROS/NetworkSetup

http://wiki.ros.org/ROS/Tutorials/MultipleMachines

XBee pro datasheet:

https://www.digi.com/pdf/ds_xbeemultipointmodules.pdf