Umeå University **June 4, 2014**
Spring 2014

Project: Park vehicle

Computer engineering and embedded systems,
15 hp

Handledare:
John Berge, Ulf Brydsten, Talib Morad

Name: Jonas Hedin
E-mail: c12jhn@cs.umu.se
Program: C

Name: Erik Berggren
E-mail: tfy11ebn@cs.umu.se
Program: C

Name: Mattias Scherer
E-mail: c12msr@cs.umu.se
Program: C

Name: Anna Jonsson
E-mail: tm11ajn@cs.umu.se
Program: C

Abstract

In this report we describe the construction of a steering system for an unmanned vehicle. The task was to construct a system that makes the unmanned vehicle follow a person automatically and it is supposed to be used to carry equipment and materials. We decided to use GPS tracking for the steering of the vehicle and we implemented a manual steering for the ease of use. The project was divided in two parts, the vehicle and a control pad used in sending the coordinates and for the manual steering. The result was that the manual steering worked perfectly but the automatic steering only works in certain conditions. To get the direction the vehicle was facing we used a compass but it was very sensitive to tilting. Another problem was that the GPS unit gave the wrong coordinates sometimes, which made the vehicle move in a different direction.

Contents

1 Problem specification **2**

2 Material and methods **3**

2.1 Control unit 3

2.2 Vehicle 4

2.2.1 Pulse Width Modulator 4

2.2.2 Acceleration 5

2.2.3 Collision protection 5

2.2.3.1 IR sensor 5

2.2.4 H-bridges 6

2.2.5 Compass 7

2.2.6 GPS 7

2.2.7 Manual steering 8

2.2.8 Automatic steering 9

2.3 Control pad/transmitter 12

2.3.1 Joystick 13

2.3.2 Slide switches 13

2.3.3 LED 14

2.3.4 Manual steering 14

2.3.5 Automatic steering 18

2.3.6 Switch between steering modes 18

2.3.7 Turn the control pad on and off 18

2.4 Communication/data transfer 19

2.4.1 Data sent from the vehicle 19

2.4.2 Data sent from the control pad 20

2.4.3 Communication algorithms 20

3 Result **23**

4 Testing **23**

4.1 Vehicle 24

4.1.1 Inside 24

4.1.2 Outside 24

4.2 Control pad 25

4.3 System tests 25

5 Limitations **25**

6 Conclusions and discussion **26**

7 References **29**

A Circuit diagram for the vehicle **30**

B Circuit diagram for the control pad **31**

C User's manual (in Swedish) **32**

D Source code **34**

1 Problem specification

The assignment was described as constructing a down-scaled prototype focusing on the steering system for a vehicle that can follow a person working in a park-environment. Two goals were specified:

1. The vehicle shall be able to to keep a constant distance from a person who is moving away from the vehicle.
2. Develop some form of manual/automatic communication which makes the vehicle stand still when the person moves towards the vehicle.

The assignment was then interpreted and written down in a more complete requirement specification which was then accepted by the client.

General requirements in descending order:

- The vehicle shall be able to to keep a constant distance from a person who is moving away from the vehicle.
- The vehicle shall stand still when the person moves towards the vehicle.
- The vehicle shall be able to be controlled manually.
- The vehicle shall not collide with walls, cats etc.
- PWM shall be used.
- The powerusage for both units shall be able to decrease with a signal from the control unit.

These requirements are the basis of the project and are the most important ones listed in the requirement specification which was formulated by us and approved by the customer (Ulf Brydsten).

A user manual for the product will also be created.

The original problem specification can be found at
http://www.moodle2.tfe.umu.se/mod/resource/view.php?id=16457

2 Material and methods

In this section the hardware, the software and the methods we have used for constructing the system are covered. All of the source code files and the circuit diagrams as well as the user manual (in Swedish) can be found amongst the attachments. A short description of the source code files (except for the header files corresponding to each source code file) follows:

Vehicle:

- `fordon.c` - Contains the logic for the vehicle: manual and automatic steering for the engines of the vehicle, algorithms for the automatic steering and communication with the control pad via bluetooth.
- `GPSParser.c` - Receives and parses data from GPS and converts longitude and latitude to doubles in decimal degree format.
- `adc.c` - Analogue to digital converter for our IR-sensors.
- `pwm.c` - Sets up two 8-bit timers and controls which compare register to use to get pulses for the duty cycles of the engines.
- `twi.c` - Receive data from the compass by using TWI communication.
- `usart.c` - Send and receive data via bluetooth.

Control pad:

- `control_pad.c` - Contains the logic for the control pad. This includes the joystick-to-engine manual steering algorithm, the GPS readings for this unit and the communication with the vehicle via bluetooth.
- `GPS_parser.c` - Receives and parses data from GPS into a string representation of each of the coordinates.
- `switch_interrupt.c` - Initializes and contains routines for the switch and on/off buttons' interrupts.

2.1 Control unit

The control unit used for the control pad/transmitter and the vehicle is a `ATMega1284P` microcontroller that is programmed using a `AVRISP mkII`-programmer. The source code is written in the programming language `C` in the developement environment `Atmel Studio`. For details regarding the implementation, we refer to the source code.

Figure 1: MCU

2.2 Vehicle

The vehicle is a Wild Thumper 6WD with six engines, one for each of the six wheels. The wild thumper is perfect for driving offroad because of the great suspension and the six engines that runs individually.

Figure 2: The Vehicle

2.2.1 Pulse Width Modulator

PWM is a way to send pulses to the engines to regulate the speed. We used phase-correct PWM which means the timer that is used increases to the maximum (255) followed by a decrease to 0, then starts over. The timers we used are using specific ports on the MCU that are set when the timer reaches a limit specified in the code. When using phase-correct PWM the ports state switches when the timers reach the limit. So if you have a high limit the ports are set for a very short time whilst when using a low limit

Figure 3: Vehicle without components

the timer spends most time over the limit which means that the port is set most of the time, and the engine controlled by that specific port goes faster.

2.2.2 Acceleration

We wanted to avoid that the engines were set to a high speed immediately — which could cause tools and other materials to fall off —, so we used a timer that increased the speed of the engine slowly. We even used it to decrease speed slowly so that the vehicle slowed down instead of just stopping.

2.2.3 Collision protection

To avoid collisions with objects we used two IR sensors which indicate the distance to the nearest object in front of them. The two sensors were placed on the front and on the back of the vehicle so that we could avoid objects when we were moving forwards and when we were moving backwards. When the values from the sensors were to high we turned off the engines and let the vehicle slow down to avoid an impact.

2.2.3.1 IR sensor

We used two Sharp GP2Y0A21 sensors in this project. The sensor itself is an analogue sensor which transmits a signal that represents the distance to the nearest object. When reading the port connected to the sensor we get a value 0-1023 which can be translated as a distance. The higher the number, the closer the vehicle is to an object.

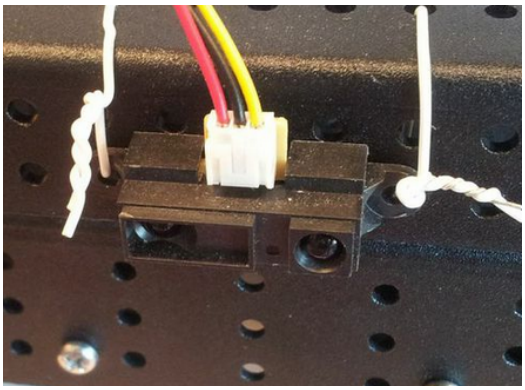

Figure 4: Distance sensor.

2.2.4 H-bridges

To power the engines we got three battery packs which each contained 6 AA batteries. Each of the pack was connected to a H-bridge that powered two engines each, one on each side of the vehicle. The inputs of the bridges were connected to the timer ports on the MCU in order to achieve that the three engines at each side run at the same speed.

Figure 5: H-Bridge.

2.2.5 Compass

We needed to know the direction the vehicle was facing, and to solve that problem we used an electric compass. The compass is of the model HMC6352 and is communicating by TWI (two wire interface), so we used the ports used for TWI on the MCU (see datasheet in the reference section or our attached circuit diagram) to connect to the compass. The compass transmits the direction the vehicle is facing with north as 0 degrees, east as 900 degrees, south as 1800 degrees and west as 2700 degrees. The value was then used for comparison with the direction for the control pad in calculating the wanted direction of the vehicle (see the subsection `Automatic steering`).

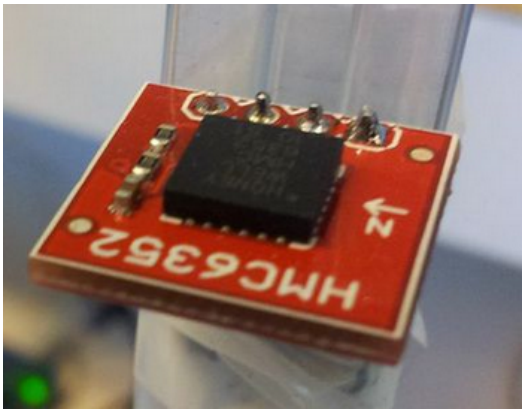

Figure 6: Compass

2.2.6 GPS

The GPS module we use is an Ultimate GPS Breakout v3 GTPA013 from Adafruit. We have to wait for the GPS to find satellites before we can get valid coordinates. The data that we get is in the form of NMEA 0183, which is a standard used in GPS-modules, an example follows below (extra return characters are inserted to avoid wrap-around, normally we would just get carriage return before every $ sign):

```
$GPGGA,092751.000,5321.6802,N,00630.3371,W,1,8,1.03,
61.7,M,55.3,M,,*75
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,
1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,
20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,16,13,28,070,17,26,23,252,,
04,14,186,15*77
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
```

```
$GPRMC,092751.000,A,5321.6802,N,00630.3371,W,0.06,
31.66,280511,,,A*45
```

The string we are interested in is the GPRMC string from where we can read the current latitude and longitude (in this case the latitude is 5321.6802 north and the longitude is 00630.3371 west, but we will get them in north and east). The coordinates' form is degrees and minutes which we have to transform before we can use them in our calculations. On the GPS there is a LED that tells you if it got a connection to enough satellites, called a fix. A fix is when the GPS is connected to 4 or more satellites and the LED indicates this by blinking every 15 seconds. When the GPS do not have a fix the LED blinks every second.

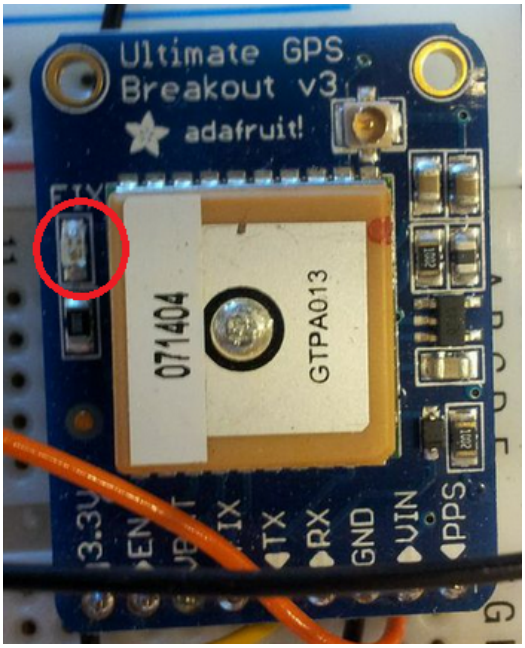

Figure 7: GPS module with the LED circled.

2.2.7 Manual steering

One of the requirements was that there would have to be a manual mode for the steering to facilitate parking and placing the vehicle according to the park worker's wish. This manual steering section describes this functionality from the vehicle's point of view.

When in manual mode, the control pad sends a byte which contains all the necessary information for manual driving. For exact format of the byte, see the `Communication/Data Transfer` section. The speed is then translated to a limit used in the timers' PWM which allows us to adjust the speed and direction of each side's engines according to the received data.

2.2.8 Automatic steering

The main requirement for this project was that the vehicle must be able to follow a person in a constant distance. This has been solved by creating an automatic mode for the steering, and this section explains how this was done.

When using the automatic steering mode, the control pad sends its GPS coordinates to the vehicle using bluetooth (see the `Communication/Data Transfer` section for more information about the transfer).

The data strings received containing the latitude and the longitude of the control pad are in the form of degrees and minutes, we want to transform them into decimal degrees so that we can use them in our calculations. This is done by parsing the data so that we get the degrees and minutes in different double variables, dividing the minutes by 60 and then adding them to the degrees. The vehicle then reads its own coordinates and transforms them to the decimal degree format as well.

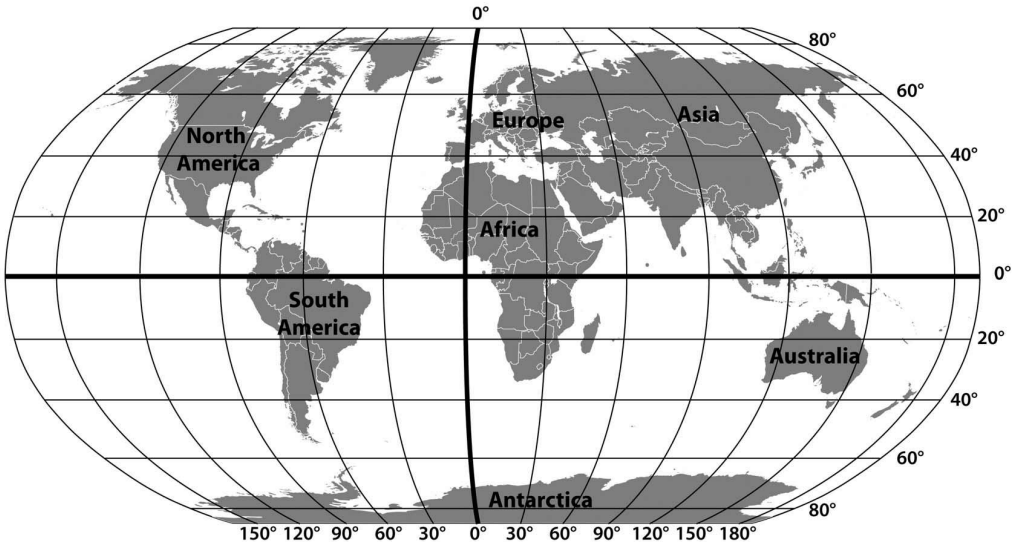

Figure 8: The GPS coordinate system.

Now we have the latitude and longitude of both units, which is all we need to calculate the *wanted angle* for the vehicle, that is, the angle in which the nose of the vehicle is pointed directly to the person in possession of the control pad. The *current angle* of the vehicle is measured using the compass, and the result is a number between 0 and 3600, which is then divided by 10, giving a result according to the picture below:

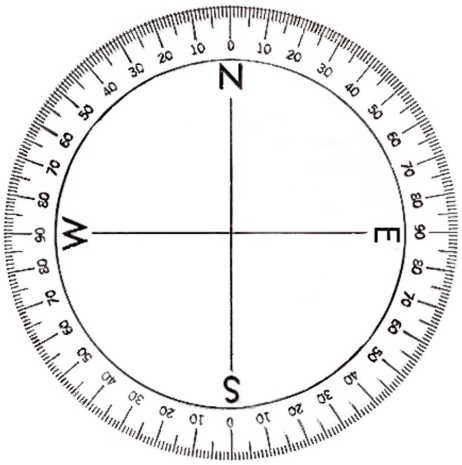

Figure 9: The compass value system.

Thus, in order to compare the current angle to the wanted angle, the wanted angle must be expressed in the same degree system. Firstly, the relationship between the coordinates of the control pad and the coordinates of the vehicle needs to be established. We define a new Cartesian coordinate system with the vehicle in the origin and by subtracting the latitude of the control pad with the latitude of the vehicle the y coordinate of the control pad in the new system based on the position of the vehicle is achieved, and the same operation on the longitudes yields the x coordinate. This follows from the properties of the values of the GPS coordinates in the first quadrant (where Sweden is, see GPS coordinate system figure), that is, increasing values on both north and east axes. Since the only thing we are interested in is the displacement angle, not the distance, we do not need to worry about converting the GPS coordinates to any other unit than degrees.

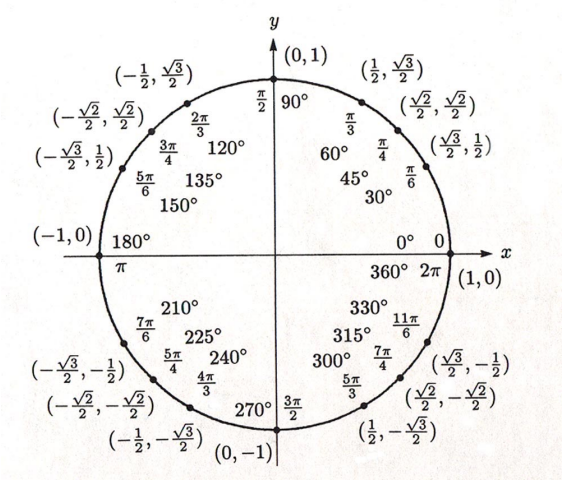

Figure 10: The unit circle representing the coordinate system with the vehicle in origo.

The wanted angle is computed using the inverse tangens function, implemented as `atan2` in the `math.h` library which returns the angle in the interval $[-\pi, \pi]$ for a point in a Cartesian coordinate system (as usual measured from the x axis and counter-clockwise), so we pass the x and y coordinates of the control pad in the new system as parameters to the function and get the angle according to the new vehicle coordinate system (see figure above).

The calculated wanted angle needs to be translated into the compass angle system, in which it is currently measured from the east axis and increases counter-clockwise instead of starting at the north axis and increase clockwise as we want it to. The formula for the wanted angle in the compass system (denoted as w) becomes

$$w = 90 - v$$

where v is the angle in the vehicle system. This is yielded by imagining a system reverse to the compass system in which an angle u is translated into the compass system by $w = -u$ and an added clockwise rotation of 90° in relation to the reversed system is expressed as $u = v - 90$ since the tilted reversed system is exactly the same as a our vehicle coordinate system. If the resulting wanted angle is negative, we simply add 360 to the value to get the result in the interval $[0, 360)$ to facilitate the comparison with the current angle. In order to get a more accurate value for the current angle, we read the compass value ten times and use the average.

The next step is to make the vehicle turn according to the new angle. To achieve this, we turn a small step at a time by setting the timer register for one side of the vehicle to a lower number than the default 255 (= the vehicle stands still) during a short period of time using delays until the angles match with a 10° accuracy in both directions. The direction of the turn is decided by comparing the wanted angle with the current angle to get the shortest turning direction. The algorithm for this is (not the exact implemented algorithm, but the same reasoning presenting all possible cases):

```
1. Calculate the difference: current angle - wanted angle.
2. If the current angle is greater than the wanted angle
and the difference is less than 180.
 2.1 Turn left.
3. If the current angle is smaller than the wanted angle
and the difference is less than -180.
 3.1 Turn left.
4. If the current angle is greater than the wanted angle
and the differece is greater than 180.
 4.1 Turn right.
5. If the current angle is smaller than the wanted angle
and the differece is greater than -180.
 5.1 Turn right.
```

When the vehicle is done turning, it starts driving forward, unless the IR sensor senses that there is an object ahead in order to prevent collisions.

This is repeated during the entire time automatic steering mode is activated until the vehicle is within a 2 meter radius from the control pad, then it stops and remains still until the control pad moves away from the vehicle. If it moves closer, nothing happens because of the vehicle still being within the range and since the IR sensors may detect the person closing up, as requested.

The distance between the vehicle and the control pad is computed using the haversine formula retreived from Wikipedia — which calulates the distance between two points on the surface of a sphere — shown below:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\left(\phi_1\right)\cos\left(\phi_2\right)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

where d is the wanted distance in kilometers, r is the earth radius in kilometers (we use the value 6362.697 km since we are in Sweden), ϕ_1 and ϕ_2 are the latitudes of the vehicle and the control pad, respectively and λ_1 and λ_2 are the respective longitudes of the vehicle and the control pad.

2.3 Control pad/transmitter

In this section the different components of the control pad are described as well as the software programmed into the `ATMega1284P` microcontroller. Apart from the components described below we also use a GPS module (see the `GPS` section) and a bluetooth unit (see the `Communication/Data transfer` section) as seen in the figure below.

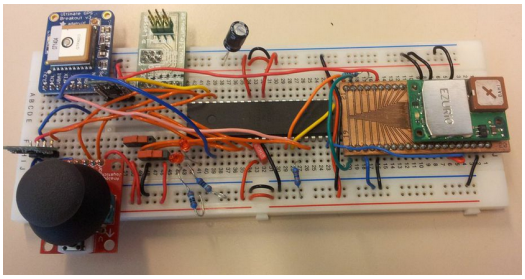

Figure 11: Control pad

2.3.1 Joystick

The analogue joystick used for manual steering has three outputs, one for each of its two axes and one for its switch which is activated by pushing down the joystick. The switch constantly emits a high signal until it is triggered, then it emits a constant low signal until the trigger is released. The switch was not used in our solution.

The axes return analogue values, these are fed to the microcontroller's internal analogue-to-digital-converter (`ADC`) so that the joystick position can be read. The analogue values range from zero to the voltage input into the V_{CC}. Both axes return half the voltage input into the the V_{CC} when the joystick is centered. When the analogue values are converted to digital values they will range from 0 to 1023, where 511 from both axes represent the joystick being centered. The digital values are then used to calculate the direction and throttle for each of the engines. How this was done is described in the `Manual steering` section.

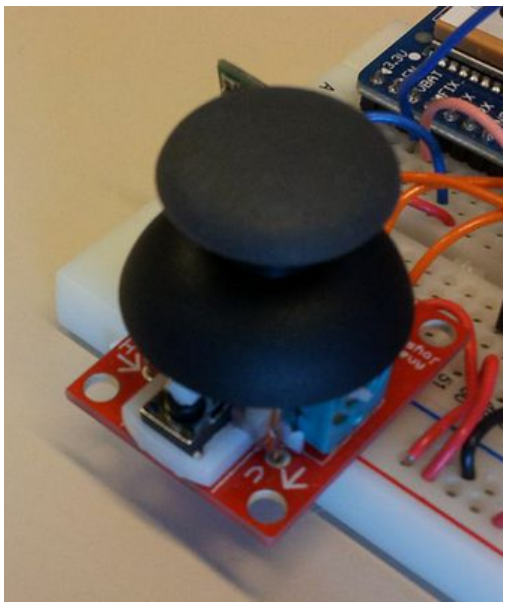

Figure 12: Joystick

2.3.2 Slide switches

In order to change the steering mode and turn on and off the control pad we have used two slide switches. When a switch is altered an interrupt is triggered and one of two possible flags (one for each switch) is set/reset. The flag is checked in the main loop which then, depending on which flag is set/reset, will turn on/off the control pad or switch between manual/automatic steering. This functionality will be described in detail later.

2.3.3 LED

We have two LED:s logically connected to the slide switches that show whether or not the control pad is on and which steer mode is used at the moment. The above LED is connected to the above switch, which is the on/off switch and the LED being lit indicates that the control pad is on. The below LED is connected to the below switch and the lit LED indicates automatic steering mode.

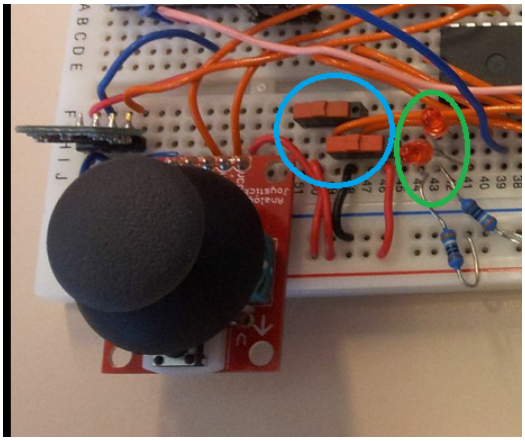

Figure 13: Switches and LED:s.

2.3.4 Manual steering

This secion describes the part of the algorithm for the manual steering done in the control pad, which is the main part of the algorithm.

By comparing the values from both the horizontal and vertical axes with fifteen constant values ranging between $0 - 1023$, the position of the joystick is converted to an integer-valued position in a two-dimensional Cartesian coordinate system where both of the axes each range from -7 to 7. We decided to use seven steps for the throttle as more steps would hardly be noticable and the throttle of each engine would fit into three bits for a more compact transfer.

A sketch outlining the thought process behind the calculations of the engines' directions and throttles based on the coordinate system defined above is shown below.

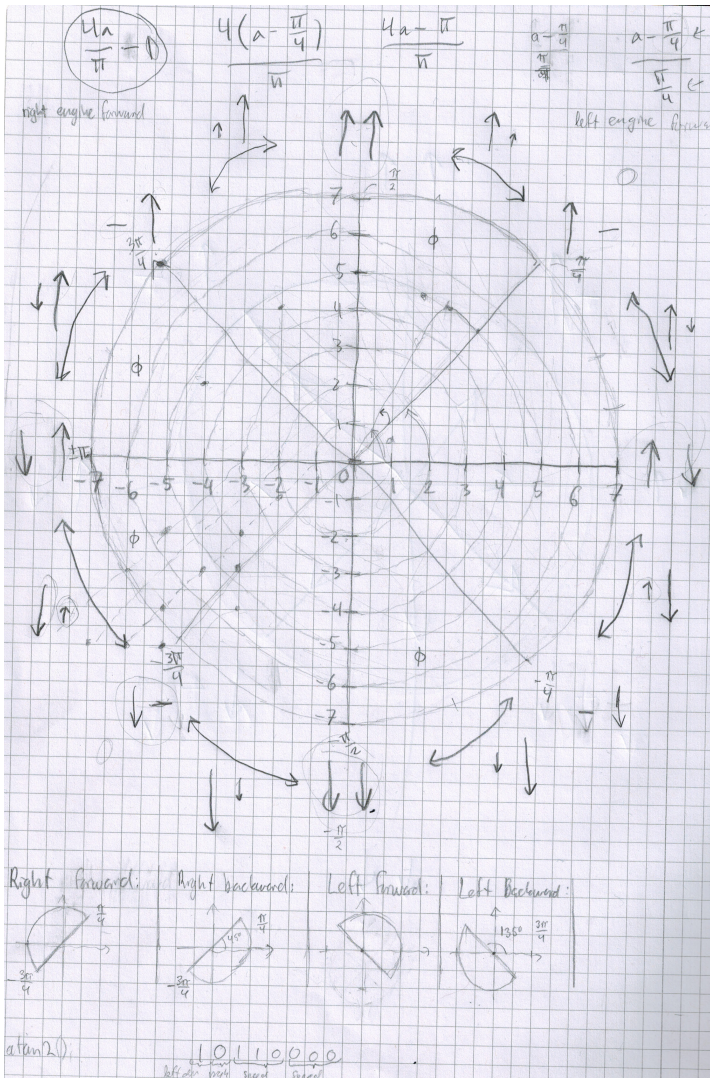

Figure 14: Joystick position translation theory sketch.

The vertical arrows represent the left and right engines' directions and relative throttles based on the position of the joystick in the coordinate system.

The directions of the engines are calculated by using the angle (v) which is calculated using the following formula:

$$\tan(v) = \frac{y}{x} \iff v = \arctan\left(\frac{y}{x}\right).$$

In our implementation, this is done by using the built-in function `atan2` from the `math.h`-library which takes the y and x coordinates as input and returns the angle in radians in the interval $[-\pi, \pi]$.

The sketch shows the left engine going forward on the interval $[-\frac{\pi}{4}, \frac{3\pi}{4}]$ and the right engine going forward on the interval $[-\pi, -\frac{3\pi}{4}] \cup [\frac{3\pi}{4}, \pi]$. This is

used to decide the directions of the engines by simply comparing the current angle to the angles above and setting the corresponding directions.

The throttles of the engines are calculated by firstly defining a main engine for each interval — the engine whose throttle changes the least in that interval, that is, the arrow in the sketch whose length does *not* change in that interval. This can be summarized as:

Main engine	**Interval**
Left	$\left[0, \frac{\pi}{2}\right) \cup \left[-\pi, -\frac{\pi}{2}\right)$
Right	$\left[\frac{\pi}{2}, \pi\right] \cup \left[-\frac{\pi}{2}, 0\right)$

The main engine's throttle is calculated by using the absolute value of the coordinate corresponding to the axis which is closest to the current position of the joystick. This is the same thing as setting the main engine's throttle to the greatest in absolute value of the x and y coordinates. Thus, the throttle of the main engine is controlled by the x-coordinate in the interval

$$\left[-\frac{\pi}{4}, \frac{\pi}{4}\right] \cup \left[\frac{3\pi}{4}, \pi\right] \cup \left[-\pi, -\frac{3\pi}{4}\right]$$

and the y-coordinate in the interval

$$\left[-\frac{3\pi}{4}, -\frac{\pi}{4}\right] \cup \left[\frac{\pi}{4}, \frac{3\pi}{4}\right].$$

Now, to get the throttle of the second engine we will have to calculate what percentage of the main engine's throttle we want to assign to the second engine depending on the angle of the joystick. The formulae used to calculate the percentage (a factor between 0 and 1) in each of the intervals

$$\left[\frac{n\pi}{4}, \frac{n\pi}{4} + \frac{\pi}{4}\right],\ n \in \{-4, -3, -2, -1, 0, 1, 2, 3\}$$

are the following:

$$\phi = \frac{v_n - v_b}{I}$$

and

$$1 - \phi$$

where v_n is the absolute value of the current angle, v_b is the base angle (the smallest angle in absolute value in the interval) for the interval that contains v_n and I is the size of the interval, which is constantly $\frac{\pi}{4}$. The current interval decides what formula is used. The rule for the formulas are that they shall

become one (1) when using the angles $\frac{n\pi}{2},\ n \in \{-2,-1,0,1,2\}$ since the engines should have the same throttles at these angles.

The factor produced is then multiplied with the main engine's throttle and as a result we get the second engine's throttle, as sought for.

The overall algorithm for setting the throttles of both of the engines is the following:

```
1. If the angle lies between 0 and pi/4.
 1.1 Calculate the factor with the formula 1-phi.
 1.2 Set the power of the left engine to the x coordinate.
 1.3 Set the power of the right engine to the power of the
 left engine multiplied by the factor.
2. If the angle lies between pi/4 and pi/2.
 2.1 Calculate the factor with the formula phi.
 2.2 Set the power of the left engine to the y coordinate.
 2.3 Set the power of the right engine to the power of the
 left engine multiplied by the factor.
3. If the angle lies between pi/2 and 3pi/4.
 3.1 Calculate the factor with the formula 1-phi.
 3.2 Set the power of the right engine to the y coordinate.
 3.3 Set the power of the left engine to the power of the
 right engine multiplied by the factor.
4. If the angle lies between 3pi/4 and pi.
 4.1 Calculate the factor with the formula phi.
 4.2 Set the power of the right engine to the x coordinate.
 4.3 Set the power of the left engine to the power of the
 right engine multiplied by the factor.
5. If the angle lies between -pi/4 and 0.
 5.1 Calculate the factor with the formula 1-phi.
 5.2 Set the power of the right engine to the x coordinate.
 5.3 Set the power of the left engine to the power of the
 right engine multiplied by the factor.
6. If the angle lies between -pi/2 and -pi/4.
 6.1 Calculate the factor with the formula phi.
 6.2 Set the power of the right engine to the y coordinate.
 6.3 Set the power of the left engine to the power of the
 right engine multiplied by the factor.
7. If the angle lies between -3pi/4 and -pi/2.
 7.1 Calculate the factor with the formula 1-phi.
 7.2 Set the power of the left engine to the y coordinate.
 7.3 Set the power of the right engine to the power of the
 left engine multiplied by the factor.
8. If the angle lies between -pi and -3pi/4.
```

```
8.1 Calculate the factor with the formula phi.
8.2 Set the power of the left engine to the x coordinate.
8.3 Set the power of the right engine to the power of the
left engine multiplied by the factor.
```

2.3.5 Automatic steering

The calculations for the automatic steering are mostly done in the vehicle — see the `Vehicle` section. The contribution to the functionality by the control pad consists only in parsing and sending the ASCII representations of the latitude and longitude values for this unit in degree and decimal minutes format to the vehicle. How it is sent is described in the `Communication/Data transfer` section.

2.3.6 Switch between steering modes

The method used for the user to be able to switch between steering modes with a slide switch is interrupt based. The switch is connected to the external interrupt port `INT2` on the microcontroller that we have set to generate an interrupt on any edge, that is, for every change in the position of the switch. This was done by setting the `EICRA` register such that the bit `ISC20` was high and the `ISC21` was low according to the information in the microcontroller data sheet. Then setting the `INT2` bit in the `EIMSK` register enabled the external interrupt on that port.

The interrupt routine only checks the global flag called `steer` that can either have the value `AUTO` or `MAN` and changes it to the other value. It also lights or turns off a `LED` that indicates which steering mode is active.

At the start of the progam, the position of the slide switch is checked and mode corresponding to that particular position is the starting mode. This is done in order to maintain the same functions of the two positions of the switch between uses.

In the main code, the steer mode is communicated to the vehicle before every main cycle in order for the correct data transfer corresponding to the current steer mode to take place.

2.3.7 Turn the control pad on and off

Similarly to how the functionality of the steering mode slide switch was achieved, interrupts were used to make on/off slide switch use possible. The

slide switch was connected to the INT1 external interrupt port on the microcontroller, and initially the register EICRA is configured to fire an interrupt on rising edge and the INT1 bit in the EIMSK is set.

When the switch is then altered, the interrupt goes off and the interrupt routine sets the power flag to OFF. In the next iteration of the main code, the flag is checked, and the control pad prepares for sleep mode by communicating the altered state to the vehicle, turning off power to all used ports, setting the ports to inputs and turning off BOD and WDT.

However, in order to wake the microcontroller from its sleep, it has to receive a low signal on an external interrupt port, so we change the INT1 interrupt to go off on a low signal instead, so that waking the microcontroller also resets the power flag to ON. Now the microcontroller is ready to go to sleep.

When it awakes, it does so in the same place in the code where it fell asleep, which means that we can communicate a wake signal to the vehicle immediately after calling the function that enters sleep mode. In this way, the synchronization of the running of the both codes is kept intact even after the control pad's microcontroller has entered and woken up from sleep mode.

2.4 Communication/data transfer

The bluetooth unit used is a EZURIO Bluetooth TRBLU23 00200 Serial Module BISM II, which has a range up to 300 metres and a high performace ceramic antenna.

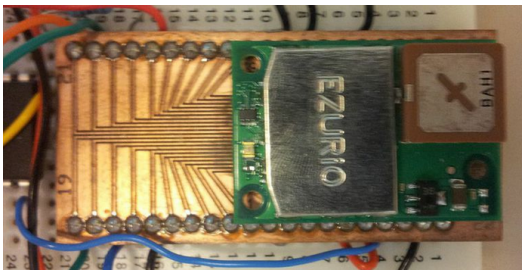

Figure 15: Bluetooth

2.4.1 Data sent from the vehicle

All data being sent from the vehicle is synchronization data, that is, data whose only use is to communicate that it is ready to receive data from the control pad unit.

2.4.2 Data sent from the control pad

The data for the manual steering that is sent from the control pad to the vehicle has the form

Data	R_v	V_2	V_1	V_0	R_h	H_2	H_1	H_0
Bit	7	6	5	4	3	2	1	0

where R_v and R_h are the directions for the left and the right engines, respectively (a one means forwards and a zero means backwards), $V_2{-}V_0$ is a number between 0 and 7 and that decides the speed of all of the three left engines and $H_2{-}H_0$ is the same but for the three right engines.

The data for the automatic steering on the other hand consists of 19 bytes in total since the latitude and longitude of the control pad are sent one ASCII character at a time. The format of the latitude data is

```
ddmm.mmmm
```

and the format of the longitude data is

```
dddmm.mmmm
```

where `d` are characters representing the degrees and `m` are the characters representing the minutes in decimal form.

2.4.3 Communication algorithms

Algorithm for the vehicle main function:

```
Algorithm: Vehicle main function

1. Initialize UART for bluetooth and GPS, timers for
steering, ADC and TWI.
2. Wait for GPS fix.
3. Do:
 3.1 Receive on/off information.
 3.2 If off mode is to be activated.
  3.2.1 Send ready-for-off-mode byte.
  3.2.2 Wait for wake byte.
 3.3 Send ready-for-steer-data byte.
 3.4 Receive steer data.
```

```
 3.5 If steer data indicates manual steering mode.
  3.5.1 Change auto steer variable to false.
 3.6 If steer data indicates automatic steering mode.
  3.6.1 Change auto steer variable to true.
 3.7 If steer mode is manual.
  3.7.1 Do a cycle of manual steering.
 3.8 If steer mode is automatic.
  3.8.1 Do a cycle of automatic steering.
 3.9 Send ready-to-receive on/off info byte.
```

Algorithm for the vehicle manual steering cycle:

```
Algorithm: A vehicle manual steering cycle

1. Send ready-for-manual-steering-byte byte.
2. Receive byte that contains manual steering info
(engine direction and power info).
3. Interpret and set the engine registers according
to the received information.
```

Algorithm for the vehicle automatic steering cycle:

```
Algorithm: A vehicle automatic steering cycle

1. Parse vehicle GPS data to a decimal degree form
double.
2. Send ready-for-control-pad-coordinates byte.
3. Receive the entire latitude in string format
(9 bytes) from the control pad.
4. Send ready-for-next-data byte.
5. Receive the entire longitude in string format
(10 bytes) from the control pad.
6. Convert the latitude and longitude to doubles
on decimal degree form.
7. Save previous distance between the two and
calculate the current distance.
8. If the vehicle is within a two meter radius
from the control pad or the vehicle is getting
further away from the control pad.
 8.1 Turn off the engines.
9. If the vehicle is outside the wanted radius
and closing up.
 9.1 Calculate the wanted angle from the control
 pad coordinates and read the current angle from
```

```
 the compass.
 9.2 Spin a few degrees at a time until the angles
 coincide.
 9.3 If the vehicle is about to collide.
  9.3.1 Turn off the engines.
 9.4 If the vehicle is not about to collide.
  9.4.1 Drive forward.
```

Algorithm for the control pad main function:

```
Algorithm: Control pad main function

1. Initialize ports, ADC, USART for bluetooth and
GPS and external interrupts.
2. Wait for bluetooth connection.
3. Wait for GPS fix.
4. Do:
 4.1 If power flag is OFF.
  4.1.1 Send power off byte.
  4.1.2 Wait for OK from vehicle.
  4.1.3 Enter sleep mode. Wake here.
  4.1.4 Send awake signal.
 4.2 If power flag is ON.
  4.2.1 Send power on byte.
 4.3 Receive ready-for-steer-data byte.
 4.4 If steer mode flag is MAN.
  4.4.1 Send a 0 for man mode.
 4.5 If steer mode flas is AUTO.
  4.5.1 Send a 1 for auto mode.
 4.6 If steer mode flag is MAN.
  4.6.1 Calculate engine data from the joystick
  position.
  4.6.2 Receive ready-for-man-data byte.
  4.6.3 Send manual steering data (1 byte).
 4.7 If steer mode flag is AUTO.
  4.7.1 Receive ready-for-auto-data byte.
  4.7.2 Parse own GPS data into latitude and
  longitude info in string format.
  4.7.3 Send latitude data (9 bytes).
  4.7.4 Receive ready-for-next-data byte.
  4.7.5 Send longitude data (10 bytes).
 4.8 Receive ready-for-next-iteration byte.
```

3 Result

The final product consists of a vehicle and a control pad with a microcontroller, a GPS module and a bluetooth unit each. The GPS modules were used to achieve the automatic steering and the bluetooth units were used for communication between the vehicle and the control pad. For the vehicle we also used IR sensors to avoid collisons, H bridges to power the engines and a compass to be able to read the direction of the vehicle, and for the control pad we used a joystick for an additional manual steering and two slide switches along with two LED:s for on/off and switching steer mode functionalities.

The manual steering works well and it is easy to use, as wanted. The joystick drives the vehicle in seven different speed states, and the vehicle turns according to the position of the joystick.

The PWM is working as intended and we used it with the smooth acceleration we implemented in order to keep the park worker's tools on the vehicle. The acceleration chosen ended up being a linear acceleration.

The IR-sensors are used to stop collisions but it does not work too well because of the range of 80 cm, sometimes the vehicle has too high of a speed to stop before the object in front of it.

It is possible to switch between manual and automatic steering at any time and even turn off the control pad to save power.

When switched into automatic steering mode in a place where the values of the GPS are correct and not reflected too much by buildings, etc., the vehicle follows the control pad with a two meter radius and stops when the control pad is closer than that.

Under the circumstances listed in the `Limitations` section, all the requirements for the product were fulfilled.

4 Testing

Continuous testing on both units as well as the entire system was done throughout the entire project. This section presents an overview of what was tested and how.

4.1 Vehicle

Tests for the vehicle were done both inside and outside, and the tests were different depending on location.

4.1.1 Inside

The first test we did were to find out if PWM worked as intended. We tried different duty cycles just to see if it worked. When the control pad was ready to send commands with the bluetooth, we tested if the vehicle followed the commands given by the control pad. This was very easy because we used practically the same code for the manual steering algorithm that we used in the previous project which suited our needs perfectly.

When we knew that the commands were sent correctly and that the vehicle did what it should, we began working on the acceleration. When it was done we did some small tests to fix the time it should take to go from standing still to maximum speed. We did several test before we decided for a suitable acceleration.

Then we connected the two IR sensors to the vehicle. We had to do many test before we found a fitting distance to start to decelerate to make sure the vehicle did not collide with the object in front or back of the vehicle.

To test the compass we set both the vehicles and the control pads coordinates to constants just to test if the vehicle spun to the right direction.

4.1.2 Outside

The first thing being noticed with the vehicle during testing outside was the problems it had with powering its engines when one of its wheels lost contact with the ground. The cause of this was assumed to be with the lack of power being sent to the engines by the batteries.

When the testing continued we noticed that the compass was very sensitive. If you tilted it the slightest you got a large error. The compass got a 50° error from just tilting it about 20 degrees. So we could only use the vehicle on a flat ground.

We also read the coordinates we got when using the automatic mode. We noticed that the coordinates we got did not line up with the correct coordinates from Google Maps. To solve this we had to use the vehicle far away from buildings and other objects that could reflect the GPS signal.

4.2 Control pad

The testing for the control pad was very streamlined since it basically only needed the same manual functionality as with the last assignment we performed, most of the testing was therefore on the automatic functionality.

Most of the testing was therefore done in order to ensure the functionality of the slide switches, which could not be fully tested until the same functionality had been implemented properly for the vehicle.

4.3 System tests

In the beginning of the project most of the problems came from our bluetooth communication. At first the problem was with synchronizing the data transfer between the bluetooth units, then when that was resolved it became apparent during testing that the current bluetooth unit had a much too short signal range. This was resolved by exchanging that unit with a new one who had 10 times the range (according to its data sheet).

A lot of time was spent with the parsing and conversion of the output sentences from the GPS unit, this was mostly because it had to be done in both the vehicle and the control pad and that the same code gave different results, which was finally resolved. We tested it by printing the parsed data using a library allowing us to use the `printf` command to write the data to a by us defined port, which came in handy when doing all data testing.

5 Limitations

We have limited our GPS tracking functionality to only work in the first quadrant in the GPS coordinate system, meaning the vehicle will be able to follow the control pad by using GPS tracking in Sweden but not in the U.S. We decided that the algoritm to allow GPS tracking to be fully functional everywhere on earth would not be too complicated for us to implement, but it would be a waste of time since this is just a prototype and will not be used anywhere else. Besides, we wanted to focus the limited time we had to get the automatic driving to work.

When the vehicle tilt, the compass can give up to $\pm 50^\circ$ heading error, which is a problem since we are developing a system to use in a tough terrain. Furthermore, we have mounted the compass on a stick above the engines to prevent any disturbance in the magnetic field, but when the vehicle is accelerating, the stick is shaking, which could tilt the compass and that

can result in a heading error. The compass we use forces us to limit the area where the automatic driving is functionable to flat surfaces. Before the vehicle requests a heading from the compass, it has to stop and wait for the stick to stabilize. This way we should eliminate a heading error but it takes more time for the vehicle to reach the control pad.

Both the control pad and the vehicle have to wait for the GPS to acquire satellites' signal and navigational data, and to calculate a position (called a fix). The GPS on the vehicle needs to get its fix first before the remote can be turned on. This is due to that the bluetooth communication must be synchronized between the vehicle and the control pad.

The vehicle has a hard time turning on rougher terrain compared to turning on asphalt. This is because every wheel might not touch the ground, and the wheels that do touch the ground do not have the power to turn the vehicle by themselves. Even though every wheel touches the ground, the weight of the vehicle can be shifted to one side, which gives traction to the wheels on that side. But the wheels on the other side won't have any traction to turn the vehicle. Also the engines receive less electic current from the batteries than they should, which makes them less powerful.

6 Conclusions and discussion

We had the following problems:

- Synchronizing bluetooth communication.
- The range of the bluetooth communication.
- The vehicle having trouble powering its engines.
- Getting the correct value from the compass.
- Finding an algorithm to consistently parse the GPS output.
- Getting the correct data from the GPS.

The synchronization of the bluetooth communication was a bit tricky at first, but after deciding to let the vehicle and control pad take turns sending bytes we did not have any more issues regarding that, except for in rare cases when they seem to get out of sync, but this is easily fixed by turning on and off the power to the both systems in the order described in the user's manual.

After resolving the synchronization problem, we discovered that the range of the bluetooth units we used at first was not enough to drive the vehicle

longer distances. That is why we changed to another bluetooth unit with 300 metres range instead of the previous 30 metres.

The problems we had with the engines not having enough power to drive forward nor backward on some surfaces could have been solved by getting a better power source. We used 26 AA batteries which might not have been the best solution. The voltage was at the right levels but the batteries did not give enough power to the engines.

To solve the problems with the compass, we could have got an accelerometer or a gyrometer to make upp for the error we had when the compass was tilted. The best solution was probably to get a new compass with three axes. From that we could have calculated the real direction. But we noticed it too late and did not have the time to incorporate a new module into the project and implement code for it.

Parsing the GPS data took a bit longer than expected since the behaviour of our program was all but what we wanted. When discovering that we had not declared enough memory for the characters but used string pointers to a non-allocated string returned from a built-in function, the issue was quickly resolved, and we learnt to read the function documentation properly.

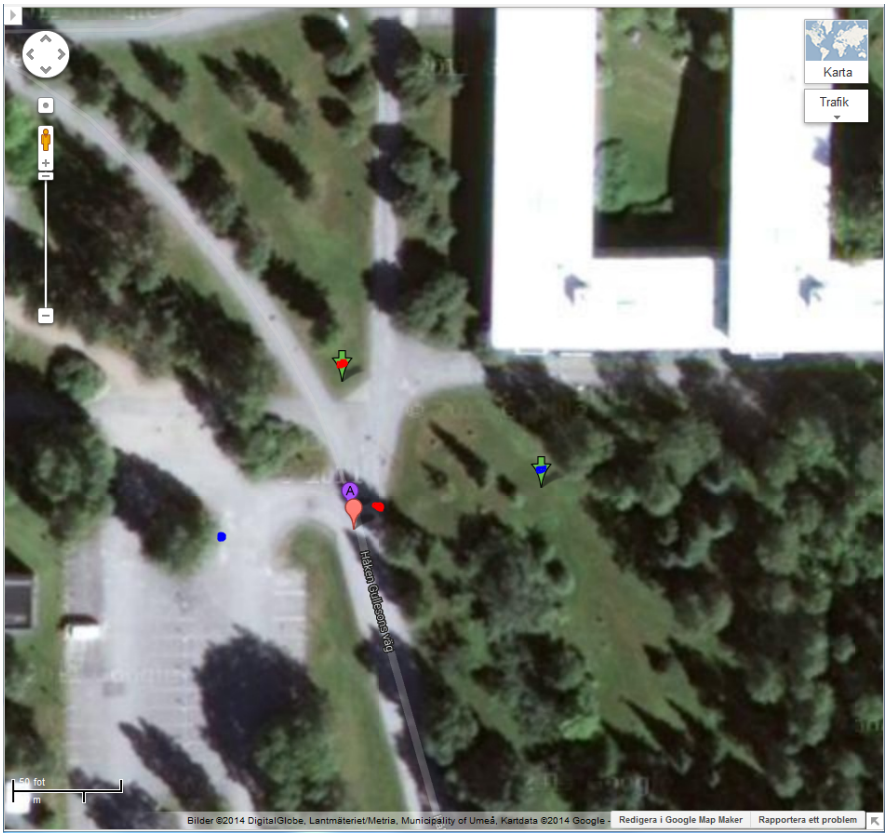

Figure 16: An example of the measured incorrect values of the GPS.

The incorrect values received from the GPS were measured outside the technical engineering building, and the result can be seen in the figure. The green arrows represent the coordinates sent to us by the GPS, the one with a red dot on it is the position for the control pad and the one with a blue dot is the position for the vehicle. The red dot represents the location the control pad was actually at, and the blue dot marks where the vehicle was at. In this particular test, the vehicle started driving north-west, which is correct according to the received values, but not according to reality.

The problem with the GPS giving us incorrect coordinates could possibly be solved by reading several values and calculate a mean value which should eliminate large errors and make the coordinates more precise. The problem that could occur is that the program could be slower. You have to get the coordinates and transform them into a different format that could be used in calculating the mean value, send those values by bluetooth and do it all over again in the vehicle before we could get a heading.

Another possible solution might be to get an antenna which could make the readings more precise. Another benefit from having an antenna is that the GPS might get the connection to the satellites quicker. The GPS unit being too close to the bluetooth unit could be another source to our problem. Since we noticed the errors too late into the project, we had not enough time to order antennas before the project was supposed to be finished.

We could have made the manual steering a bit better if we sent the signals we got from the joystick directly to the vehicle instead of transforming it to the format we used before sending it. We would have to send more bytes, but in the end, we should have got a more precise steering that worked directly from the joystick.

The acceleration of the vehicle used with the manual steering could at times be mistaken for a delay and made the overall manual steering of the vehicle a bit trickier. Perhaps we should have allowed the speed to increase and decrease a bit more rapidly.

7 References

Atmel, 8272E–AVR–04/2013, *8-bit Atmel Microcontroller with 16/32/64/128K Bytes In-System Programmable Flash: ATmega164A/PA/324A/PA/644A/PA/1284/P.*

Ezurio Ltd., 2005-2006, http://www.manuallib.com/file/2590398, *EZURIO Bluetooth TRBLU23 00200 Serial Module BISM II Manual.*

Pololu, http://www.pololu.com/product/2123, *Pololu 5V Step-Up/Step-Down Voltage Regulator S7V8F5.*

Sparkfun, https://www.sparkfun.com/tutorials/272, *Thumb Joystick.*

Adafruit, http://www.adafruit.com/products/746, *Adafruit Ultimate GPS Breakout v3.*

Sparkfun, https://www.sparkfun.com/products/7915, *Compass Module - HMC6352.*

Freescale, http://www.freescale.com/files/analog/doc/data_sheet/MC33887.pdf, *H-bridge.*

Sparkfun, https://www.sparkfun.com/products/11056, *Wild thumper 6WD.*

Wikipedia, http://en.wikipedia.org/wiki/Haversine_formula, *Haversine formula*, hämtad 2014-05-26.