Gesture based user interface

Project report in II2302 Sensor Based Systems

Jatesada Borsub
Michal Tomaszuk
Tanvir Saif Ahmed

II2302 Sensor Based Systems, 7,5 HP
Examiner: Mark T Smith

KTH
School of Information and Communication Technology
164 40 Kista, Sweden

Abstract

In today's society, people tend to forget about certain, small tasks that need to be done. Tasks like turning off the lights when leaving the room, turning off the stove, etc. Some people, for example the elderly or the disabled, are not able to easily do those tasks, as moving around their living spaces may be difficult. To be able to easily multitask is a possible solution to these problems.

This project report describes a system for gesture based user interface, and how it was constructed in terms of hardware and software architecture.

The system consists of a transmitter and a receiver part, where the transmitter part acts as a hand-held device for the user to interact with the receiver part of the system. These parts of the system include ultrasonic sensors and radio modules able to communicate with each other by sending different types of signals.

The software part of the system is heavily relying on MATLAB, which translates receiver data into figures and plots them, showing the shapes user has drawn with the transmitter. These shapes may include simple polygons such as a triangle or a square, but even more complex, three-dimensional shapes are possible.

Keywords

multitasking, disabled people, gesture based user interface, transmitter, receiver, ultrasonic sensors, radio modules, communication, MATLAB, shapes

Content

1 Introduction

This part of the report describes the system and its application.

1.1 System

The constructed system is a gesture based user interface, and consists of four ultrasonic sensors and two radio modules. One ultrasonic sensor and one radio module make up the transmitter part and the other three ultrasonic sensors along with the second radio module make up the receiver part of the system.

The transmitter part acts as a hand-held device for the user, which then communicates with the receiver part. Different types of signals are being transmitted from the hand-held device, in particular radio and ultrasonic waves. As radio waves and sound travel at different speeds, it is possible to calculate the distance between the transmitter and the receivers using the signals' time of arrival. As radio waves arrive almost instantly (traveling at literal speed of light), they are used to start a timer. An ultrasonic wave is sent at the same time as the radio wave, and as soon as it reaches a receiver, the timer is stopped. Only the time of the ultrasound wave's arrival changes noticeably when the hand-held device is moved, while the time for the radio to arrive to the receiver part of the system stays nearly the same – such small distances are hardly noticeable with the speed of light.
In order to get the pulses needed to trigger the time measurement, a digital processing part has been added to the system, as ultrasonic receivers output noisy signals, making it difficult for the time measurement to start and stop exactly when the signals arrive. By using Arduino Leonardo, the raw output signals from the ultrasonic sensors are received and then translated to squared signals through software, which can output low and high values and make sure that the time measurement gets activated and deactivated.

To determine the gesture that the user is making, the time values of the ultrasonic sensor receivers are calculated continuously on Arduino MEGA 2560 and then sent through the serial port to an external computer running MATLAB, where the distances are calculated and later used to calculate the positions. These positions will determine the gesture that the user is making and plot the positions on a 3D graph.

1.2 Application

The system can be used to allow the user to run certain tasks by making a gesture with the hand-held device, which is then detected up by the receivers. For example, drawing a square can turn the lights on, while drawing a triangle can start music playback.

This application could be useful for people who are disabled (and for example have problems moving around) or people that want to multi-task. The application can solve the problems that these type of people are facing by giving the ability to do certain tasks in a quicker time and from a long distance.

2 Architecture

In this part of the project report, a description of the software and hardware architecture that was used to implement the project is given. This includes a description of the hardware parts that were used, hardware schematics, software flowcharts, block diagrams of the full system and other information that describes the implementation of the system and application.

2.1 Hardware

2.1.1 Arduino

In order to control external devices, sensors and integrated circuits used in the implementation, two microcontrollers were used. These are Arduino MEGA 2560 and Arduino Leonardo, both based on Atmega-processors [1].

2.1.1.1 Arduino MEGA 2560

The Arduino MEGA 2560 (see figure 1) is a microcontroller that consists of 54 digital and 16 analog I/O ports, where each port can deliver a current of 40 mA with a voltage of 5 V. The microcontroller has up to 256 kB of storage (where 8 kB is used by the bootloader) and its clock speed is 16 MHz [2]. For this project, the Arduino MEGA 2560 was mainly used for controlling the signals coming from the ultrasonic receivers and the radio receiver module in order to measure the times of arrival, which are later sent through the serial port to an external computer running MATLAB.

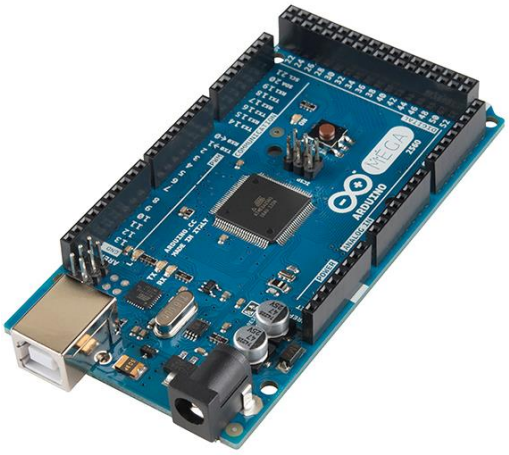

Figure 1: The figure shows the microcontroller Arduino MEGA 2560 [7].

2.1.1.2 Arduino Leonardo

The Arduino Leonardo (see figure 2) is a microcontroller that consists of 20 digital and 12 analog I/O ports, where each port can deliver a current of 40 mA and a voltage of 5 V. It has 32 kB of storage (where 4 kB is used by the bootloader) and its clock speed is 16 MHz [3]. For this project, the Arduino Leonardo was mainly used for processing the received signals coming from the ultrasonic receiver sensor by translating the raw signals to squared signals and send the squared signals to Arduino MEGA 2560 for triggering the time measurement.

Figure 2: The figure shows the microcontroller Arduino Leonardo [8].

Since several signals had to be processed simultaneously, three Arduino Leonardos was used, one for each ultrasonic receiver sensor.

2.1.2 Ultrasonic sensor kit

An ultrasonic sensor is a distance sensor that sends out sound waves and then reads the pulse that it receives from the object. By this operation, the time from the transmitter to receiver can be determined and used to calculate the distance [4]. There are several types of ultrasonic sensors which includes transmitter transducer (see figure 3) and receiver transducer (see figure 4) being assembled on the same circuit or the transmitter transducer and receiver transducer being separated on their own circuits and communicating with each other by sending a certain frequency. For this project, separate ultrasonic kits have been used, which are transmitter transducer and receiver transducers, being separated on their own circuits. This is because the response time needs to be when the transmitter sends a pulse to the receivers by deactivating the time measurement, in order to find the distances and then positions for scanning the gesture. Since the gesture that the user can be three-dimensional, three ultrasonic receiver kits were required.

Figure 3: The figure shows the ultrasonic transmitter kit with the transducer [9].

These ultrasonic kits consists of ultrasonic transducer, amplifiers, timer-ICs, transistors and can send and receive a frequency of 40 kHz. The ultrasonic kits can also be constructed from scratch, but since many circuits for the receiver and transmitter that are found on the internet vary a lot regarding the components used, the ultrasonic kits was bought instead.

Figure 4: The figure shows the ultrasonic transmitter kit with the transducer [10].

2.1.3 Radio module

Radio modules are small electronic devices that can be used to transmit and receive radio signals, which travel at the speed of light. These type of devices are mostly used in applications for the purpose of communicating wirelessly between two endpoints by sending and receiving radio waves [5]. For this project, two radio modules have been used, which is the radio transmitter (see figure 5) and the radio receiver (see figure 6). The radio modules are used to calculate the time for the ultrasonic signal to reach from source to destination by activating the time measurement in Arduino MEGA 2560, since the radio signals travels faster than ultrasound signals.

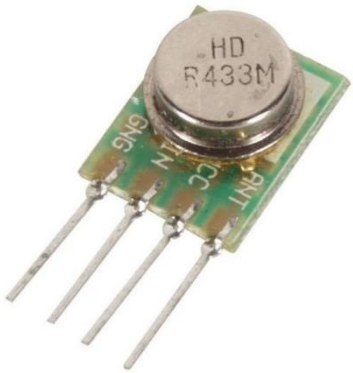

Figure 5: The figure shows the radio transmitter module [11].

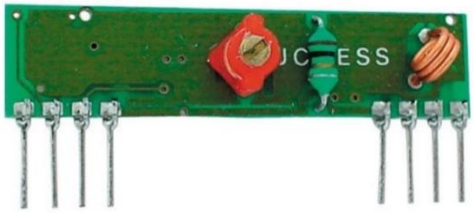

Figure 6: The figure shows the radio receiver module [12].

2.1.4 Hardware description

In this part of the architecture, a description of the system is given in terms of hardware. This includes the transmitter and the receiver part of the system.

2.1.4.1 Transmitter part

2.1.4.1.1 Ultrasonic transmitter

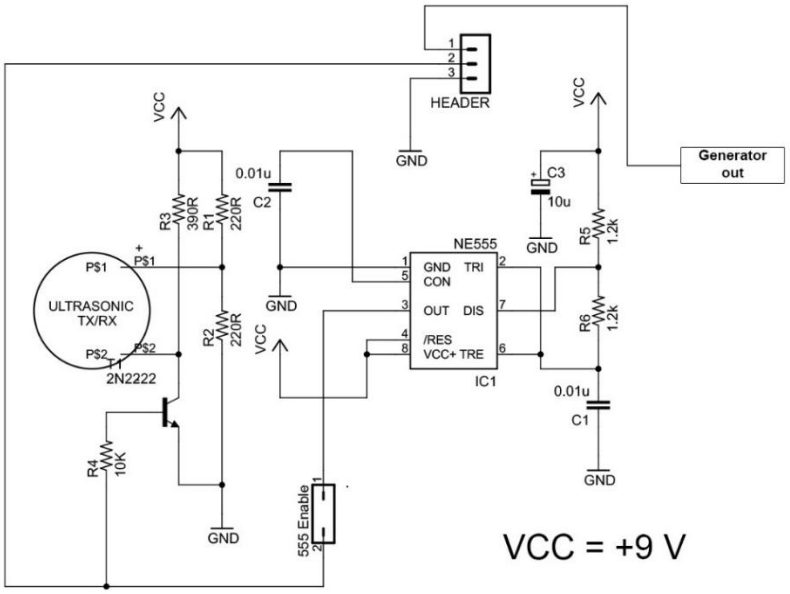

Figure 7: The figure shows the schematic for the ultrasonic transmitter kit [13].

The ultrasonic transmitter kit (see figure 7) includes a 555 timer IC, a transistor and an ultrasonic transducer. The 555 timer IC generates 40 KHz square pulse signals, which is an ultrasound frequency. These signals are sent to the transistor for amplifying the signals in order to gain sufficient power for sending out the signals to the ultrasonic receivers by the ultrasonic transmitter transducer.

2.1.4.1.2 Radio transmitter

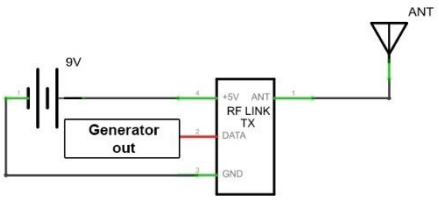

Figure 8: The figure shows the schematic for the radio transmitter module.

The radio transmitter module TX433N (see figure 8) consists of 5V, GND, DATA and ANT pins, where the DATA pin is used to generate the 7 Hz radio signal with a carrier frequency of 433.92 MHz.

These signals are sent to an antenna for sending out the signals to a radio receiver module.

2.1.4.1.3 System frequency generator

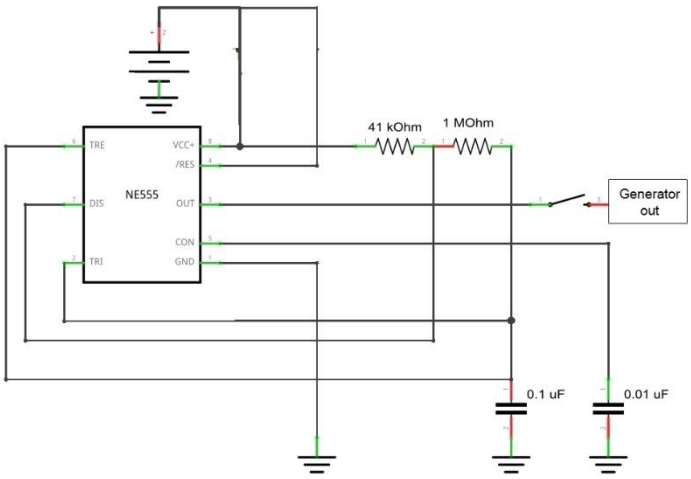

Figure 9: The figure shows the schematic for the system frequency generator.

The system frequency generator (see figure 9) includes a 555 timer IC to generate 7 Hz square pulse signals as the data signal of the system. These signals are sent to the ultrasonic transmitter and the radio transmitter. The data signal is carried by 40 kHz and 433.92 MHz carrier signals which are generated by the ultrasonic transmitter and the radio transmitter respectively. The data signals coming from the ultrasonic transmitter and the radio transmitter are later sent to the receiver part of the system.

2.1.4.2 Receiver part

2.1.4.2.1 Ultrasonic receiver

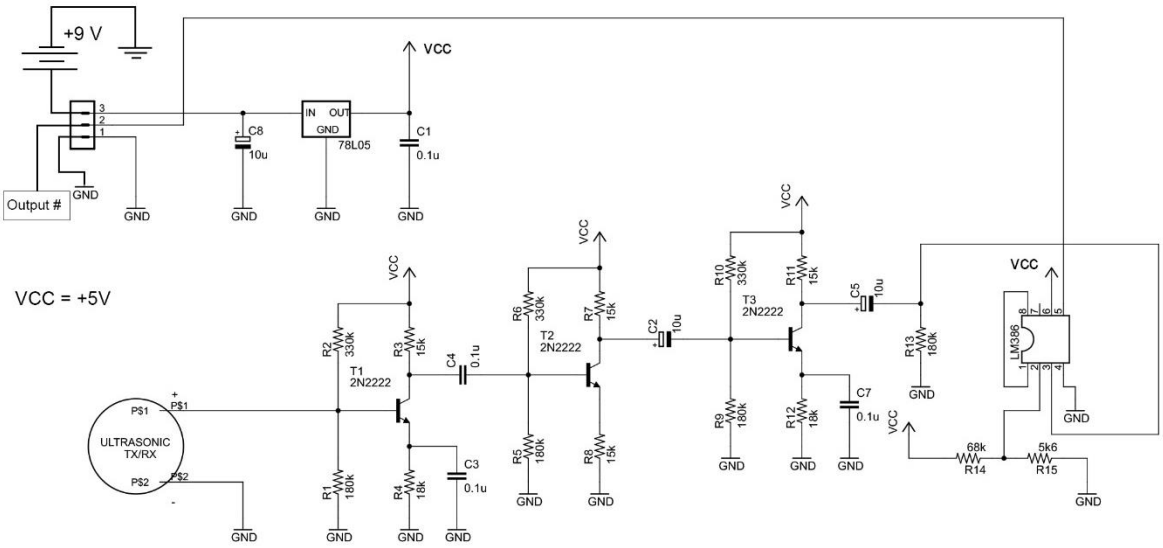

Figure 10: The figure shows the schematic for the ultrasonic receiver kit [14].

The ultrasonic receiver kit (see figure 10) includes an ultrasonic amplifier IC, transistors and an ultrasonic transducer. The ultrasonic transducer receives the 7 Hz data signal which is carried by the 40 KHz ultrasonic signal.

The data signal is sent to the transistors and the ultrasonic amplifier IC for pre-amplifying the signals in order to gain sufficient power for sending out the signals to the ultrasonic signal amplifier.

2.1.4.2.2 Radio receiver

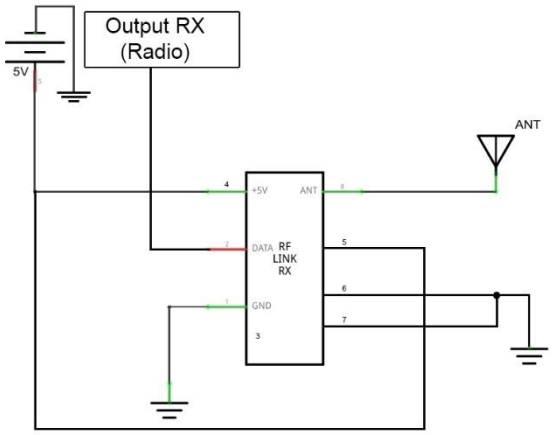

Figure 11: The figure shows the schematic for the radio receiver module.

The radio receiver module RX433N (see figure 11) consists of 5V, GND, DATA and ANT pins, where the ANT pin is used to receive the 433.92 MHz signal from the antenna. DATA pin is used to send the signal to Arduino MEGA 2560 through a digital pin in order to activate the time measurement.

2.1.4.2.3 Ultrasonic signal amplifier

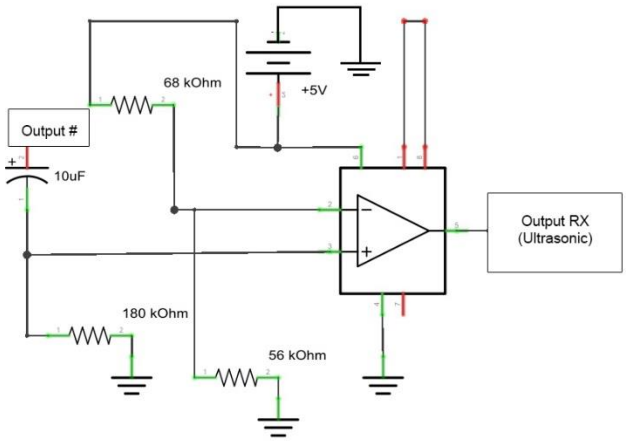

Figure 12: The figure shows the schematic for the ultrasonic signal amplifier.

Since the voltage of pre-amplified ultrasonic signals are very low, it is impossible for the digital input of the Arduino Leonardo to read these signals. In order to make it possible for Arduino Leonardo to read these signals, the signals have to be gained by ultrasonic signal amplifier LM386 (see figure 12). After the signals are amplified, the result will give 5 volts pulse signal at the frequency 7 Hz. These amplified ultrasonic signals will be sent to the Arduino Leonardo through digital pins for digital processing, in order to eliminate noises of amplified ultrasonic signals.

The digital processed ultrasonic signals are then sent to Arduino MEGA 2560 in order to stop each three arrival times (deactivating the time measurements) of the three ultrasonic receivers which are located in different positions. This will show the information of the time of arrival.

2.2 Software

2.2.1 Software description

In this part of the architecture, a description of the system is given in terms of software. This includes the flowcharts for the software programs that was used for the system.

2.2.1.1 Digital processing

Ultrasonic receivers used in the system output a 5V-signal ('HIGH'-level) by default. As soon as a pulse arrives, the receiver outputs 0 V for a split moment (just a few microseconds), then returns to 5V, then goes down to 0V for a longer while. The first time the signal goes to 0 V and output a ‘HIGH’ value to the timing system (Arduino MEGA 256) needs to be picked up.

To achieve this, every receiver is connected to an Arduino Leonardo, which continuously reads its input. As soon as a 'LOW'-level is read, the circuit outputs a 'HIGH'-level and holds it for 50 µs, so that the timing circuit has time to pick it up.

The reason to do this is to take as much of the load off the timing system, as it has to be as precise as possible. This could be optimized, for example by using a single Leonardo for all three of the receivers, but using one for each was easier and quicker, while creating less sources of error.

For the programming source codes of the Arduino software regarding digital processing, see appendix 1.

2.2.1.2 Measurement of time

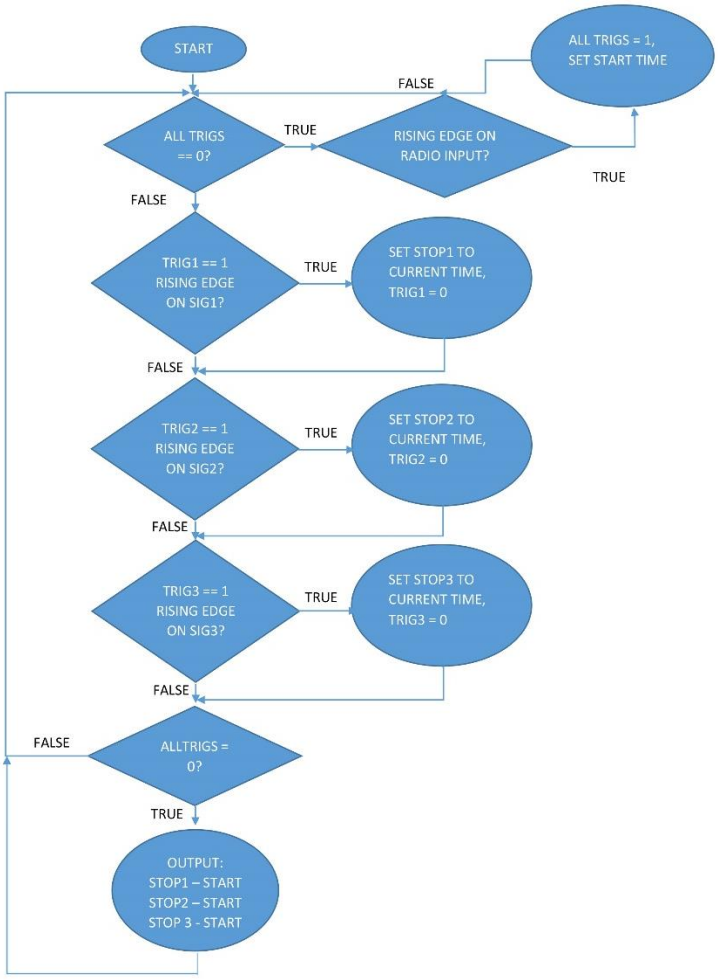

Figure 13: The figure shows the flowchart for the measurement of time.

To time the signals with the radio wave, all three digitalized pulses along with the radio wave were piped into an Arduino MEGA 2560 (see figure 13). Each signal piped to Arduino MEGA 2560 had two flags – **old_value** and **current_value**, and the three ultrasonic signals also got their **trigger** flags. This totals 11 binary flags, and all of them were saved in a single short integer named **sigs**, and all the flags were initiated to 0 in the beginning.

The program starts by looking at the **triggers**. If all of them are 0, the program reads the radio signal, and compares it to its **old_value**. If the **old_value** is 0 and **current_value** is 1, then there is a rising edge, otherwise **current_value** is copied into **old_value** and it starts over. In the case of a rising edge, all three **triggers** are set to 1, and the current processor time is saved (in us) in the long unsigned integer **start**. After that, it starts over, but this time we can move down the flowchart.

For every of the **triggers**, the corresponding signal is read. If the signal has a rising edge (checked in the same manner as for the radio signal), we set its **trigger** to 0 and save current processor time in the corresponding long unsigned integer **stop**.

We do this until all three **triggers** are set to 0 (which means that we have measured the time for all three signals), in which case we output the three resulting times (**start-stop** for each signal) to the serial port, to be read by the next part of the software. After that, the program loops and the radio wave step starts over, as all **triggers** are 0 again.

This part of the code has been heavily optimized, and relies on simple comparisons using bitwise-logic, to make the workload as light as possible, since timing has to be as exact as possible.

For the programming source codes of the Arduino software regarding time measurement, see appendix 2.

2.2.1.3 Capturing of the gesture

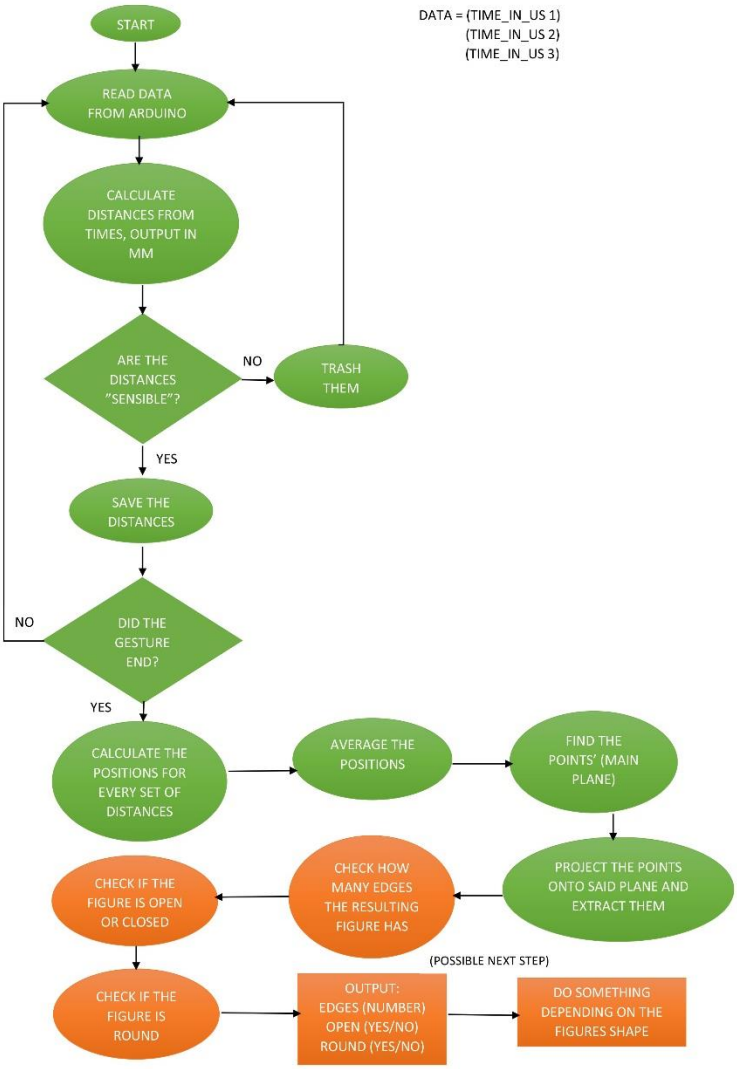

Figure 14: The figure shows the flowchart for capturing the gesture.

The next part of the software is run on a separate PC, using MATLAB.

First, the time data is read from the Arduino MEGA 2560 using the serial port (see figure 14). This gives us three times, one for each receiver, which are then placed in a vector.

The times are given in microseconds, and since speed of sound in air is 340 m/s or 0.34 mm/µs, we can simply multiply the times by 0.34 to get the distances to each receiver in millimeters.

Then a quick reality check is made – if any of the distances is too high or too low (thresholds are set manually, but it is assumed that we are at least 500 mm from any of the receivers, and at the most 3000 mm), this set of distances is discarded, as it is clearly an erroneous value, and a new set is read from the serial port, otherwise the distances are saved in a separate array.

Acquisition of times and translation to distance are repeated until a gesture is finished. To finish a gesture, the user has to break the line of sight between the transmitter and the receivers, which results in erroneous values coming into MATLAB – those values are, as mentioned previously, discarded, after which new values are read, but if a certain amount of errors are encountered in a row (the threshold is currently set to 5), the gesture is assumed to be finished, and the loop exits.

The next step is to translate each set of distances to point positions. This is done by solving the following system of equations:

$$x^2 + y^2 + z^2 = d1^2$$

$$(x - x1)^2 + y^2 + z^2 = d2^2$$

$$(x - x2)^2 + (y - y2)^2 + z^2 = d3^2$$

Here, x, y and z are the coordinates of the point in question, and d1, d2 and d3 are the distances to each of the receivers. It is assumed that the receivers are placed in a local coordinate system at positions (0,0,0), (x1, 0, 0) and (x2,y2,0). The system of equations uses the principle of trilateration, where a point's position can be determined using its distance to three other, known points. Since three points are always sharing a mutual plane, there will be two possible solutions, but as the transmitter is always on one side of the receivers, this is not a problem, as one of the solutions can simply be discarded. Once all the sets are translated into points, we can plot the figure, an example of which can be seen in figure 17. Here, the resulting shape that represents a square is noisy, so the points are averaged to get finer results.

The next step is figure recognition. Here, we assume the gestured figure is two-dimensional, and thus we will only be checking a two-dimensional shape. First, we find the figure's midpoint and move it to the point of origin (0,0,0), and then calculate the figure's normal vector. We calculate rotational matrices, to rotate the figure onto the xy-plane, and then we set all z-values to 0, which results in the figure being effectively projected onto the xy-plane.

The next steps are purely theoretical, as none of them have been fully completed. To recognize the figure gestured, we check how many edges the shape has. This is done by looking at each point's distance to the figure's midpoint, then looking at that distance's derivative – any discontinuity means that we have just hit a “corner” (as in a square's corner), and looking at the amount of corners we can discern the amount of edges. This works great for clean figures, as well as ones with small amounts of noise, but the ones gotten from the actual gesturing are too noisy, and the script sees more edges that there should be.

Then we check the “openness” of the figure – simply put, if the start and end points are too far apart, the figure is open, but if they are close, it is some kind of a polygon. Then, the last and least viable step of recognition was to look at how “round” the figure is - if the distances to the midpoint are more or less similar, the figure is quite round – but this could barely work at all, due to the amount of noise. With these three values, we could discern what kind of a shape the gesture was – for example, edges=4, openness=0 and roundness=0 means we got a square, while 3,0,0 is a triangle. This did, unfortunately, not work. If it did, we could then simply choose an action to complete based on the figure, for example the shape square will turn on the lights.

For the programming source codes of the MATLAB software regarding capturing the gesture, see appendix 3, 4, 5, 6, 7, 8, 9 and 10.

2.3 Design

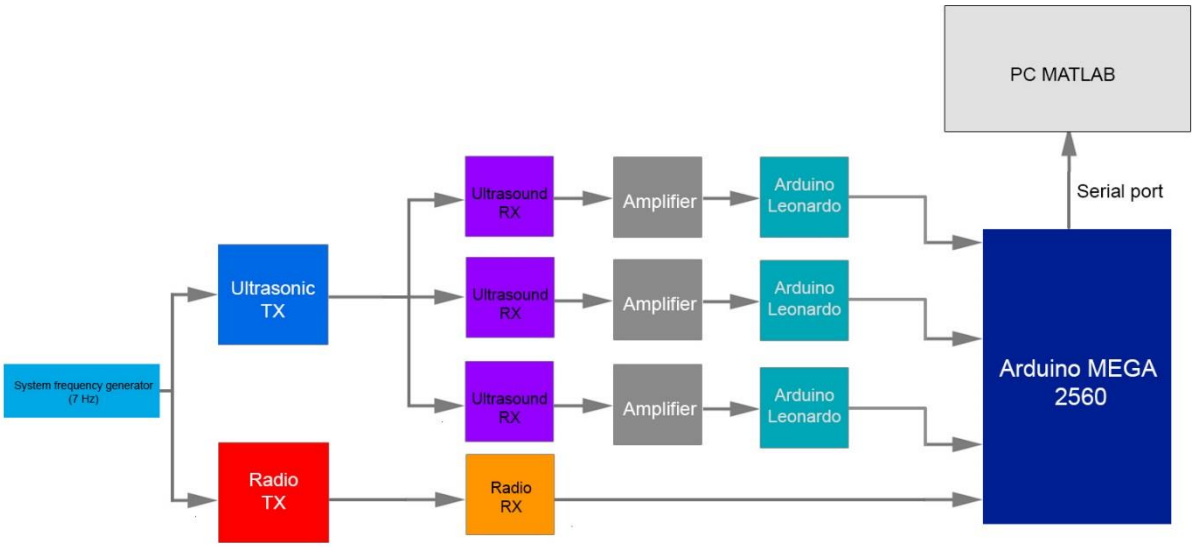

Figure 15: The figure shows block diagram of the complete design for system.

A complete block diagram (see figure 15) is given above for the system that shows the parts structure and the interaction between the parts. The left part of the block diagram shows the transmitter part of the system, which is the ultrasonic transmitter, radio transmitter and the system frequency generator. These are the parts that sends radio waves and ultrasound waves in a certain frequency. The middle part of the block diagram, shows the receiver part of the system, which is the radio receiver, ultrasonic receiver and the amplifier. These are the parts that will receive the radio waves and ultrasound waves, which are digital processed through Arduino Leonardo and the time response of these signals are then calculated in Arduino MEGA 2560. The time values are then sent to MATLAB through serial port.

3 Data measurements and results

In this part of the project report, a collection of data measurements and results is given in form of values and figures.

3.1 Stationary condition – measurements

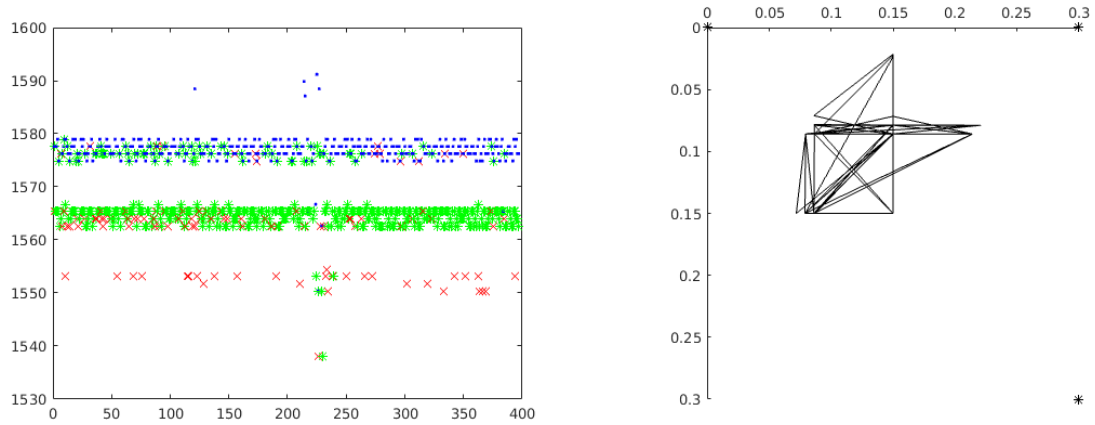

Figure 16: The figure shows a distribution graph and a 2D graph for the three receivers location in a stationary condition.

The above figure (see figure 16) shows first a distribution graph of the measurements from the three receivers (receiver one = blue points, receiver two = green points, receiver three = red points) and the other graph shows the location of the three receivers in 2D. The above results are given in the condition when the transmitter part of the system (handheld device) is stationary. Some of the errors that are shown in the distribution graph are neglected because these errors occurs from disturbance in the environment i.e. people pass through between the transmitter and the receivers.

3.1.1 Accuracy

Looking at the figure 16, the true values of the receivers can be determined as:

$$Receiver\ 1\ (blue\ points) = 1576\ mm$$
$$Receiver\ 2\ (green\ points) = 1565\ mm$$
$$Receiver\ 3\ (red\ points) = 1564\ mm$$

Each of the receivers arithmetic mean are calculated to:

$$Receiver\ 1\ (blue\ points) = 1576{,}55\ mm$$
$$Receiver\ 2\ (green\ points) = 1565{,}7\ mm$$
$$Receiver\ 3\ (red\ points) = 1560{,}47\ mm$$

The accuracies are then calculated to:

$$Receiver\ 1\ (blue\ points) = |1576 - 1576{,}55| = 0{,}55\ mm$$
$$Receiver\ 2\ (green\ points) = |1565 - 1565{,}7| = 0{,}7\ mm$$
$$Receiver\ 3\ (red\ points) = |1564 - 1560{,}47| = 3{,}53\ mm$$

The results shows that three different accuracy values is given, where receiver 1 is the most accurate and receiver 3 is the least accurate.

3.1.2 Precision

Looking at the figure 16, the range of distances for the receivers can be determined as:

$$Receiver\ 1\ (blue\ points) = 1575 - 1579\ mm$$
$$Receiver\ 2\ (green\ points) = 1562 - 1578\ mm$$
$$Receiver\ 3\ (red\ points) = 1550 - 1565\ mm$$

The precisions are then calculated to:

$$Receiver\ 1\ (blue\ points) = 4\ mm$$
$$Receiver\ 2\ (green\ points) = 16\ mm$$
$$Receiver\ 3\ (red\ points) = 15\ mm$$

The results shows that three different precision values are given, where receiver 1 is the most precise and receiver 2 is the least precise.

3.1.3 Resolution

The range of data that will be accurate for the system is:

$$500 - 3000\ mm, which\ is\ equal\ to\ 2500\ discrete\ values$$

This gives:

$$2500 \rightarrow \frac{log(2500)}{log(2)} = 11{,}28 \approx 11 bits$$

The resolutions are then calculated to:

$$Receiver\ 1\ (points) = 11 - \frac{log(4)}{log(2)} = 11 - 2 = 9\ bits \rightarrow 2^9 = 512 \rightarrow \frac{2500}{512} \approx 5\ mm$$
$$Receiver\ 2\ (points) = 11 - \frac{log(16)}{log(2)} = 11 - 4 = 7\ bits \rightarrow 2^7 = 128 \rightarrow \frac{2500}{128} \approx 20\ mm$$
$$Receiver\ 3\ (points) = 11 - \frac{log(15)}{log(2)} = 11 - 3{,}9 \approx 7\ bits \rightarrow 2^7 = 128 \rightarrow \frac{2500}{128} \approx 20\ mm$$

The results shows that three different resolution values are given, where receiver 1 has the lowest resolution and receiver 2 and 3 has the highest resolution.

3.2 Output shapes - results

When the user is drawing the shapes, the output results is given below in terms of three graphs (see figure 17, 18 and 19). The first graph is the distribution graph of the measurements in mm from the three receivers (receiver one = blue points, receiver two = green points, receiver three = red points). The second graph shows the location of the three receivers in 2D or 3D. Since the figure is too noisy, the points are averaged in order to give a better result of the shape, which can be shown in the third graph.

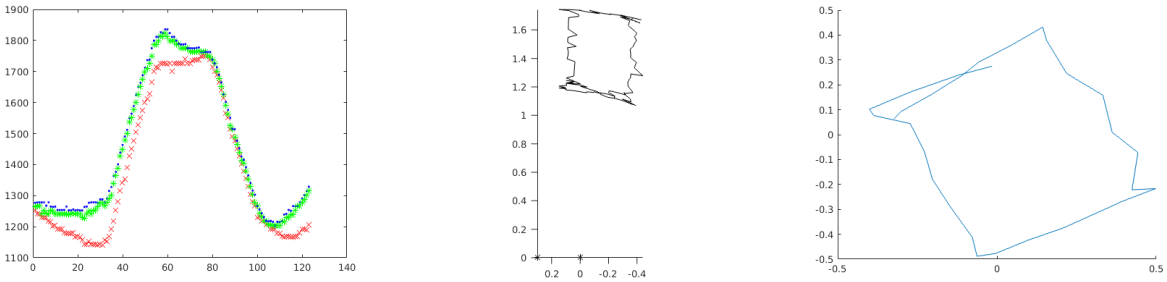

Figure 17: The figure shows a distribution graph, 2D graph and an average shape graph for the three receivers location in a square condition.

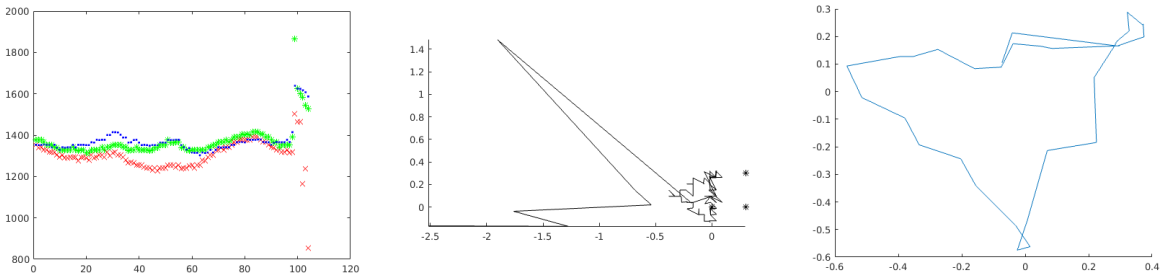

Figure 18: The figure shows a distribution graph, 2D graph and an average shape graph for the three receivers location in a triangle condition.

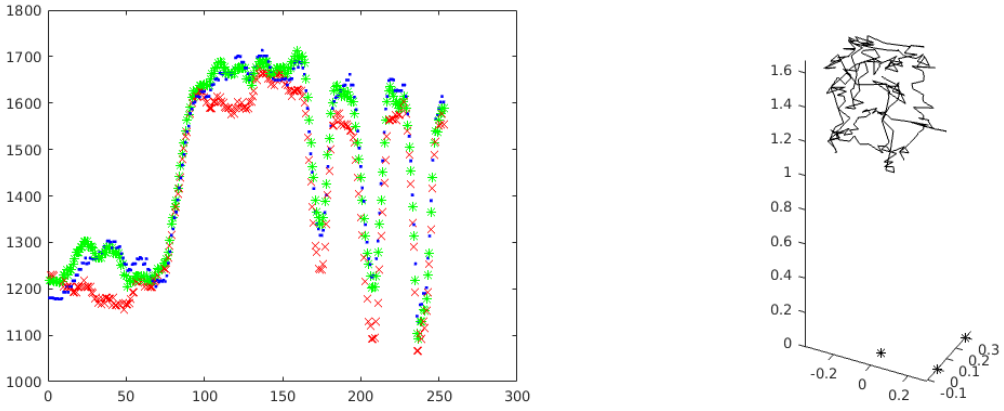

Figure 19: The figure shows a distribution graph and a 3D graph for the three receivers location in a squared box condition.

4 Conclusion

In this part of the project report, conclusions about the design is given by all of the project members.

4.1 First conclusion

The gesture based user interface works based on the time of arrival theory. In order to get the information of the time of arrival in a three dimension coordinate system, the system includes a radio transmitter and a radio receiver which are electromagnetic wave devices for starting system timer and includes an ultrasonic transmitter and 3 different location ultrasonic receivers which are sound wave devices for stopping system timer. The time of arrival information is processed by MATLAB program. The process gives several geometric shapes of the gestures. In addition, signal amplifiers and digital processor have been implemented in order to increase the robustness of the system.

From measurement analysis, we can see that the accuracy of the system is 3.5 mm, the precision of the system is 16 mm and the resolution of the system is 20 mm. These characteristics are acceptable in order to work with hand gesture detection application. However, when we have tested the system in the real situations, we found that the system has been disturbed by other system which were used ultrasonic transducers or radio transducers. This problem would be account as the subject of improvement for the next research.

The total power consumption of the system is around 2.5W (the system is operated at 9 V). Most of power of the system is consumed by Arduino microcontrollers. The total cost of the system is around 1500 Kr.

The system quite large for commercial using. The system should be improved the design to reduce in both scale and power consumption. In addition, the system is needed to improve in the way that regards the elimination of ultrasound and electromagnetic noises. The range of the radio wave that we use is 433.92 MHz. This range has disturbed by other computer devices a lot i.e. laptops, microcontrollers and mobile phones. I will change the range of radio transducer which is not a widespread used range in order to avoid the radio communication traffic. This will give the better result for accuracy and precision of the system. The digital signal processors which are Arduino microcontrollers should be changed to other devices which consume less power and cheap. The concept of the time of arrival theory, which uses electromagnetic wave devices and sound wave devices is not preferred to be changed, since this method has been working quite well in this system.

It is possible that people would buy the product, because the tele-operating application is useful for disabled people and is convenient for multitask working. However, this prototype is too large. If the system has improved its size and its performance such as accuracy, precision and power consumption, people would need this device.

The commercial leap motion sensor [6] which offers three dimension gesture control costs 600 kr. Then, people might spend their money for this product around 500 – 1000 kr.

Written by: Jatesada Borsub

4.2 Second conclusion

Looking at the design that has been made, the gesture based user interface is working on an acceptable level. Using the time of arrival theory, it is possible to find the response times from the three ultrasonic receivers and use it to find out the positions of the transmitter part of the system. This is done by sending out different signals, which will calculate the time by activating the time measurement by radio signal and deactivate the time measurement by ultrasound signal. The output results of the system shown on the MATLAB program, which represents the shapes that the user is making, is also acceptable and have been more robust due to subsystems that have been added to the system, such as signal amplifier, digital processor and system frequency generator.

The data measurements that have been received from the output shows that the maximum accuracy is 3.57 mm, the maximum precision 16 mm and the maximum resolution is 20 mm. These results are decent for a prototype that have been constructed in this project. The application could work better if the system is in an environment where there is surrounding systems containing ultrasonic sensors or radio modules, since these can disturb the system and lead to wrong results that the output system gives.

The performance of the system is working on an acceptable level. There are not any noticeable delays or other types of sources that causes the system to perform slow in terms of erroneous results (such as signals giving out wrong time responses). This is due to several microcontrollers that are used, which has a high clock frequency and processes the data quickly. However, the power consumption is quite high since there are many external devices that needs to be operated all the time. This includes the microcontrollers, ultrasonic sensors, radio modules, signal amplifers, etc. Overall cost of the system is around 1500 kr, where most of the cost comes from the microcontrollers and the sensors and modules used.

The design of the system is possible for commercial using, but can be better in terms of the size of the system. Since a large battery box is powering the ultrasonic receivers, it can be problem to carry the system around and place it in a different place. Also, the power consumption is high and the design is quite sensitive to other electronic devices around.

The design can be improved by using other types of ultrasonic sensors and radio modules that operates on different frequencies. Since the radio modules sends and receives radio signals with a carrier frequency of 433.92 MHz that is also used by other electronic devices such as laptops, mobile phones, etc., disturbance to the system can occur. The disturbance can also be caused by the frequency 40 kHz that the ultrasonic sensor is using, which can be due to surrounding systems where ultrasonic sensors generate the same frequency range. Using different frequency ranges will give better results for the output system in terms of accuracy, precision and resolution. The noise could be minimal by adding more subsystems to the system such as high pass filter, which can filter out a certain type of frequencies that the external devices is operating in. In order to decrease the power consumption, the amount of microcontrollers used for digital signal processing should be decreased to one microcontroller which can handle simultaneous tasks and consumes less power. This will also decrease the cost.

It is possible for people to buy the system, since this will help the disabled people and other types of users to do certain tasks in a quicker time and from a long distance. However, since this is just a prototype in its early stage, the system needs to be improved in a lot of areas in order to make the product popular.

Due to the high sensitivity to noise that the prototype has, a reasonable price for the product should be around 300 kr. In the future, when a lot of areas have been improved, for example noise reduction, better accuracy, less consumption, the price for the next version of the product should be around 600 – 1000 kr.

Written by: Tanvir Saif Ahmed

4.3 Third conclusion

The project has, unfortunately, not achieved all its goals. It is now possible to use the transmitter to draw a shape recognizable by a human observer, but the software is not able to recognize it on its own, which means that the “interface” is not really working.

The system has multiple other drawbacks. One of the most important issues is that it is very sensitive to outside noise and interference. For example, if any other group was experimenting with ultrasound nearby, even if they used frequencies far outside our ranges, then our system would not work, as it would not pick up the correct ultrasonic pulses. Another problem was that the transmitter had to always point directly towards the receivers – otherwise the pulses would be lost, and only delayed echoes would be picked up, resulting, once more, in erroneous values. This property has actually been used in the project – by breaking the line of sight, hugely erroneous values were generated, and after getting five such values in a row, a gesture was deemed “completed”, instead of having to look for any other way to stop the program.

The size of the transmitter is not satisfactory either. If a gesture based system was to be used, it would first of all need to be a convenience, not at hindrance that requires holding huge transmitters and waving them around. If it was possible to scale it down, possibly to the size of a ring placed on a person's finger, then it could be much more useful. There are already transmitter-less systems on the market that offer gesturing as a means to communicate (for example Microsoft's Xbox camera attachment, Kinect), but many of them use complicated, costly means of reading the user's gesture and need clear line-of-sight along with good lightning. If it was possible to get around the need of having a clear line of sight with the ultrasonic receivers (which is theoretically possible, through looking at echoes and so on, but this is a very, very complicated matter), then out system would have a clear advantage.

Yet another problem was the cost. This is, as mentioned previously, a proof of concept, a prototype at best – but with the cost of around 1500kr and a good amount of costly manhours of work required, it is not feasible as something to put on the market, especially with the product not being really complete.

If these problems were to be fixed – less sensitivity to errors, no need for a clear line of sight, the program being able to actually determine a gesture and complete an action based on the gesture – then, along with a more reasonable price tag (possibly around 500-1000kr), I could clearly see people wanting to buy it – it is after all something that I myself would like to have.

Written by: Michal Tomaszuk

References

Information references

[1] Arduino, "Arduino Boards",

http://playground.arduino.cc/Main/AVR
Visit date: 2016-01-02.

[2] Arduino, "Arduino MEGA 2560",
https://www.arduino.cc/en/Main/ArduinoBoardMega2560
Visit date: 2016-01-04.

[3] Arduino, "Arduino Leonardo",
https://www.arduino.cc/en/Main/ArduinoBoardLeonardo
Visit date: 2016-01-03.

[4] Ultrasonic sensor, "Ultrasonic sensor measurement principles",
http://www.sensorcentral.com/photoelectric/ultrasonic01.php
Visit date: 2016-01-07.

[5] Radio module, "RF modules & solutions",
http://www.futureelectronics.com/en/wireless-rf-radio-frequency/rf-modules-solutions.aspx
Visit date: 2016-01-07.

[6] Leap motion, "Leap motion sensor"
http://www.gizmag.com/leap-motion-gesture-control-sensor/22644/
Visit date: 2016-01-16.

Image references

[7] Arduino MEGA 2560,
http://www.robotshop.com/media/files/images2/rb-ard-33-1.jpg

[8] Arduino Leonardo,
https://cdn.sparkfun.com//assets/parts/6/9/5/3/11286-01.jpg

[9] Ultrasonic transmitter kit,
http://www.engineeringshock.com/uploads/1/2/5/2/12523657/s712034898615967039_p136_i1_w533.jpeg

[10] Ultrasonic receiver kit,
http://www.engineeringshock.com/uploads/1/2/5/2/12523657/s712034898615967039_p311_i1_w1100.jpeg

[11] Radio transmitter module,
http://www.kjell.com/se/image/Product_223309sv/full/1

[12] Radio receiver module,
http://www.kjell.com/se/image/Product_223307sv/full/1

[13] Ultrasonic transmitter circuit,
http://www.engineeringshock.com/40khz-ultrasonic-transducer-transmitter-kit.html

[14] Ultrasonic receiver circuit,
http://www.engineeringshock.com/40khz-ultrasonic-transducer-receiver-kit.html

Appendix

Programming source codes

Appendix 1: C code for digital signal processing of the ultrasonic receiver signals.

```
int digitalPin = 3;
int digitalOutput = 4;

int val = 0;


void setup()
{
  pinMode(digitalOutput, OUTPUT);
}

void loop()
{
  val = digitalRead(digitalPin);

  if(val == 1)
  {
    digitalWrite(digitalOutput, LOW);
  }
  if(val == 0)
  {
    digitalWrite(digitalOutput, HIGH);
    delay(50);
  }

}
```

Appendix 2: C code for time measurements of the three ultrasonic receivers.

```
void setup ()
{
  Serial.begin (9600);
  pinMode (2, INPUT);
  pinMode (3, INPUT);
  pinMode (4, INPUT);
  pinMode (5, INPUT);
}

short sigs = 0; // bit0-bit10: r_in, r_old, 1_in, 1_old, 2_in, 2_old, 3_in,
3_old, trig1, trig2, trig3

unsigned long start, stop0, stop1, stop2, stop3 = 0; // timing variables

long int period = 100000; // period of radio wave in us, 1/f (set somewhat
lower, e.g. if period is 250000, set it to 200000)

void loop()
{
  if ( (sigs & 1792)==0 )                                  // if all
triggers == 0
  {
    sigs = (sigs & ~1) | digitalRead(2);                 // reset r_in and
read in new val

    /* check if radioOld==0 and radioIn==1 (rising edge) AND 200ms have
passed */
    if ( (sigs & 3)==1 && (micros()-start)>period )
    {
      start = micros();                                  // start timer
      sigs = sigs | 1792;                                // set trigger bits
    }

    sigs = (sigs & ~2) | ( (sigs & 1) << 1 );            // set radioOld to
radioIn value
  }

  else
  {
    sigs = (sigs & ~84) | digitalRead(5)<<6 | digitalRead(4)<<4 |
digitalRead(3)<<2; // reset in-signals and throw in new vals
    stop0 = micros();
    if ( (sigs & 256) && (sigs & 12)==4          // trigger1 & rising edge
on sig1
    {
      stop1 = stop0;                             // stop timer1
      sigs = sigs & ~256;                        // reset trig1
    }

    if ( (sigs & 512) && (sigs & 48)==16 )       // trigger2 & rising edge
on sig2
    {
      stop2 = stop0;                             // stop timer2
      sigs = sigs & ~512;                        // reset trig2
    }

    if ( (sigs & 1024) && (sigs & 192)==64 )     // trigger3 & rising edge
on sig3
    {
```

```
      stop3 = stop0;                                  // stop timer3
      sigs = sigs & ~1024;                            // reset trig3
    }

    if ( !(sigs & 1792) )
    {
      String pr = "";
      pr += stop1-start;
      pr += ", ";
      pr += stop2-start;
      pr += ", ";
      pr += stop3-start;
      pr += ";";
      Serial.println(pr);

    }

    sigs = (sigs & ~168) | ( (sigs & 84) << 1 );          // reset old
values and set them to what was read in before
  }
}
```

Appendix 3: MATLAB code for the main wrapper.

```
clear all, close all, clc

pause(3)

samps = 400;
recdist = 300;
dist_coeff = 1;

err_cutoff = [1000 2000 5];

fclose(instrfind);
delete(instrfind);
arduino = serial('/dev/ttyS111','BaudRate',9600);
fopen(arduino);

vtot = zeros(3,samps);
snd1 = [sin(1:+.6:400), sin(1:+.7:400), sin(1:+.4:400)];
snd2 = [sin(400:-.6:1), sin(400:-.7:1), sin(400:-.4:1)];
sound(snd2);
pause(1)

pos1 = zeros(3,samps);
figure(1)
subplot(2,1,1)
plot3(recdist*[0 1 1],recdist*[0 0 1],[0 0 0],'*b'), hold on

errs = 0;
errs_tot = 0;

c = 1;
format shortG
for i=1:samps
    getdata = fgetl(arduino);
    v = str2num(getdata)'*0.34;
%    if mod(i,samps/2)==0
%        sound(0.5*snd1);
%    end
    if errs>err_cutoff(3)
        break;
    end

    if (max(abs(v)) < err_cutoff(1)) || (max(abs(v)) > err_cutoff(2))
        %c = c-1;
        errs = errs+1;
        errs_tot = errs_tot +1;
        disp([errs_tot i v'])
    else
        vtot(:,c) = v;
        errs = 0;
        pos1(:,c) = dist2pos(dist_coeff*vtot(:,c),recdist);

        if c>1
            %figure(1)
            subplot(2,1,1)
            plot3([pos1(1,c) pos1(1,c-1)],[pos1(2,c) pos1(2,c-
1)],[pos1(3,c) pos1(3,c-1)],'k'), hold on, axis equal
        end
```

```
        %figure(2)
        subplot(2,1,2)
        plot(c,v(1),'rx'), hold on
        plot(c,v(2),'g*')
        plot(c,v(3),'b.')
        pause(0.1);
        c = c+1;
    end
end


sound(snd1);
format short

L = c-err_cutoff(3);
vtot2 = vtot(:,1:L);

%inds = ones(1,L);
%for i=1:3
%    inds(find(vtot(i,:)>1500)) = 0;
%    inds(find(vtot(i,:)<900)) = 0;
%end
%vtot2 = vtot(:,find(inds));

%L = length(vtot2);

pos2 = zeros(3,L);
%plot3(recdist*[0 1 1],recdist*[0 0 1],[0 0 0],'*k'), hold on, axis equal
for c=1:L
    pos2(:,c) = dist2pos(dist_coeff*vtot2(:,c),recdist);
end
%plot3(pos2(1,:),pos2(2,:),pos2(3,:),'.-');

v2 = rework(pos2,L);

v2 = shape_filter(v2,30);

acq_fig = figure_recog(v2);

figure()
plot3(v2(1,:),v2(2,:),v2(3,:)), hold on
%plot(v2(1,:),v2(2,:));
```

Appendix 4: MATLAB code for finding the figures main plane and re-rotate.

```
function rerotd_p = rerot(points)

x = points(1,:)-mean(points(1,:)); y = points(2,:)-mean(points(2,:)); z =
points(3,:)-mean(points(3,:));

n = [sum(x.^2) sum(y.*x) sum(z.*x);sum(x.*y) sum(y.^2) sum(z.*y);sum(x.*z)
sum(y.*z) sum(z.^2)]\[-1;-1;-1];
n = n/norm(n);
v = [0;0;1];

n1 = [0; n(2); n(3)]; n1 = n1/norm(n1);
s1 = norm(cross(v,n1)); c1 = dot(v,n1);
Rt1 = [1 0 0; 0 c1 -s1; 0 s1 c1];

n2 = Rt1*n;
n2 = [n2(1); 0; n2(3)]; n2 = n2/norm(n2);
s2 = norm(cross(v,n2)); c2 = dot(v,n2);
Rt2 = [c2 0 s2; 0 1 0; -s2 0 c2];

Rt = Rt2*Rt1;
rerotd_p = Rt*[x;y;z];
%rerotd_p = rerotd_p(1:2,:);
```

Appendix 5: MATLAB code for the edge count of the figures.

```
function [amnt, roundness, edges] = edge_count(points)

L = length(points(1,:));

k = zeros(1,L);

for i=1:L
    k(i) = points(1,i).^2+points(2,i).^2;
end
dk = sign(diff(k));
%dk = sign(dk);

for i=2:length(dk)-1
    if dk(i-1)==dk(i+1)
        dk(i) = dk(i-1);
    end
end
dk=diff(dk);
edges=(find(dk>0));
amnt = length(edges);

k = k/max(k);
roundness = sum((k-min(k)).^2)/abs(min(k))<1;
```

Appendix 6: MATLAB code for the shape filter.

```
function filt_p = shape_filter(points,order)

[s1,s2] = size(points);

filt_p = zeros(s1,order);
for j=1:s1
    i_start = 0;
    i_step = round(s2/order);
    for i=1:i_step:s2
        i_start = i_start+1;
        i_end = i+i_step-1;
        if i_end > s2
            i_end = s2;
        end
        filt_p(j,i_start) = mean(points(j,i:i_end));
    end
end

%filt_p = filt_p(:,1:end-1);
filt_p(:,end) = points(:,end);
```

Appendix 7: MATLAB code for openness figures.

```
function state = opennes(points)

x1 = points(1,1); y1 = points(2,1);
x2 = points(1,end); y2 = points(2,end);

dist_m = 0;
s = length(points(1,:))-1;
for i=1:s
    dist_m = dist_m + 1/s*sqrt((points(1,i)-points(1,i+1))^2 +
(points(2,i)-points(2,i+1))^2);
end

state=0;

if sqrt((x1-x2)^2+(y1-y2)^2)>(4*dist_m)
    state=1;
end
```

Appendix 8: MATLAB code for rework.

```
function n_points = rework(points,order)

n_points = shape_filter(points,order);
n_points = rerot(n_points);
n_points = rescaler(n_points);
```

Appendix 9: MATLAB code for rescaling.

```
function res_p = rescaler(points)

p_t = points;

[~,s] = size(p_t);
fs=factor(s);
p_t = shape_filter(points,s/fs(1));
dc = sqrt(sum(p_t(1:2,:).^2));
[~,ind] = max(dc);
%dc = p_t(1:3,1);

c = p_t(2,ind)/norm(p_t(1:2,ind)); s = sqrt(1-c^2);

p_t = points;
p_t = -[c s 0; -s c 0; 0 0 0]*p_t;

x = -min(p_t(1,:))+max(p_t(1,:));
y = -min(p_t(2,:))+max(p_t(2,:));
%z = -min(p_t(3,:))+max(p_t(3,:));

res_p(1,:) = p_t(1,:)/x;
res_p(2,:) = p_t(2,:)/y;
res_p(3,:) = p_t(3,:);%/z;
```

Appendix 10: MATLAB code for figure recognition.

```
function f = figure_recog(points)

[edges,rndns,~] = edge_count(points);
opens = opennes(points);

f = [opens,edges,rndns];
```