CODEN:LUTEDX/(TEIE-5419)/1-35(2019)

Industrial Electrical Engineering and Automation

Virtual Leash

Device-Less Remote Parking Based on Ultrasonic Sensor Detection

Pontus Strömberg

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Virtual Leash – device-less remote parking based on ultrasonic sensor detection

Pontus Strömberg

LUNDS UNIVERSITET

Division of Industrial Electrical Engineering and Automation

Division of Industrial Electrical Engineering and Automation
Lund University
Box 118
SE-221 00 LUND
Sweden

Lund 2019

Abstract

In today's car industry more and more focus is put on autonomous driving and driving assistance. There are functions being developed that are supposed to assist the driver as well as functions that will be fully autonomous. Parking is an area where car companies are developing a lot of functions related to driver assistance and autonomous driving. One of the main reasons for this is that a lot of people think it is difficult to park their cars. Therefore, functions related to parking assistance have a high customer value. And in order to make a good function related to parking assistance and autonomous parking the function must be easy to use.

In this thesis work a proof of concept was made for the use of ultrasonic sensors in an automatic parking function, where you can remotely control the movement of a car in parking situations without any remote device. Instead of controlling the car with a phone or a key, the car was to recognize the user as the driver and allow him/her to lead it in and out of parking spaces, by having it following his/her movement.

The purpose of the thesis work was to investigate whether or not ultrasonic sensors could be used to track the movement of the user outside of the car, to be able to authorize the user as the driver. If the ultrasonic sensors could detect a specific pattern of movement from the user, the user would be allowed to remotely operate the vehicle. A simulation model was created that was able to read ultrasonic data from a test car and evaluate the movement of the user. This model could read data from a recorded log and in real time in a test car. Since the purpose was to evaluate the performance of the ultrasonic sensors, other signals related to the function was mocked in the simulation.

Acknowledgements

This master thesis was written at Volvo Cars. I would like to thank everyone in Team Concept for their help, especially Mathivalavan Gunaseelan, Andreas Abramsson and Nenad Lazic.

Contents

1. Introduction

The idea of autonomous driving is very appealing to a lot of people. Functions that operates your car fully or partly autonomous are being developed at a higher pace than ever before. Parking is an area where a lot of functions have been developed the last few years, and a lot of effort is put on making parking more and more autonomous. The biggest reason for this is because there are a lot of parking scenarios that people find difficult, and therefore parking assist functions have a high customer value. The problem with a lot of functions within parking assistance is that you have to either be in the car, or tell the car what to do with some remote device. Instead of the user controlling the car with a device, we would like to make the car recognize the user and follow its movement.

1.1 Virtual Leash

Virtual Leash is a function in Remote Automatic Parking. Like other Remote Parking functions, Virtual Leash is operated without being in the vehicle. What distinguishes Virtual Leash from other automatic parking functions is that instead of having the user control the car with some device, the car recognizes the driver with the use of ultrasonic sensors and follows the driver's movement. The function operates similarly for park in and park out, but there are a few minor differences. [1]

Park out

In a Park Out scenario the user uses his/her phone to actuate the Virtual Leash function. The phone sends the request to the car which starts to scan for the phone through Bluetooth. Once the car has detected the phone's Bluetooth signal, the car starts to scan for the key. This is done through the car's Passive Entry system, which detects the key on a distance of approximately 1,5 meters around the car. As soon as the car detects the key that is paired with the phone that was used to start Virtual

Leash, the ultrasonic sensors located in the front or rear, depending on the parking situation, start to scan.

To be able to remotely operate the car, the user has to be recognized by the car as the driver. This is achieved when the car is detecting the key and the ultrasonic sensors are detecting an object that is moving from one side of the car to the other, in the front or the back, depending on parking direction. By walking from one side of the car to the other, while the car detects the key, the user is confirmed as the driver of the car. This will be referred to as the initiation sequence in the report, see figure 1.

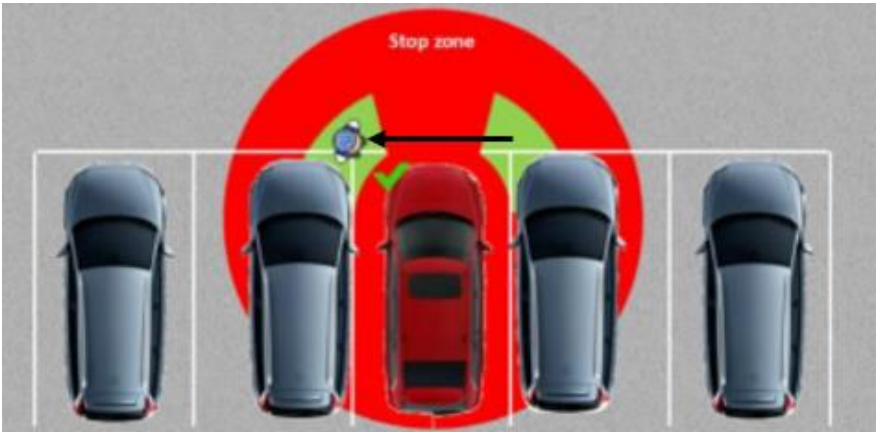

Figure 1. User has moved from one side of the car to the other and completed the initiation of the Virtual Leash.

When the initiation sequence is completed the car confirms this to the driver by driving forwards a few centimeters. Then the user is allowed to walk the car out of the parking spot as long as no other object is detected in front of the car, see figure 2. This will be referred to as the Keep Alive sequence in the report.

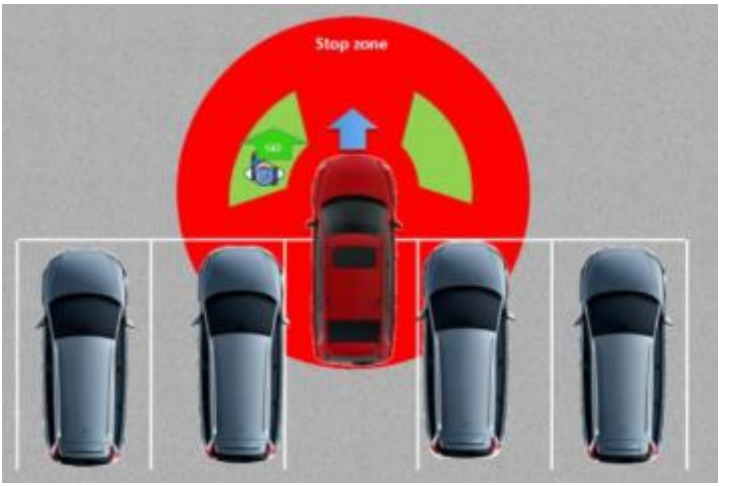

Figure 2. The car is following the driver out of the parking spot.

The car will stop, and the Virtual Leash will end once the car has completed the predefined parking maneuver or if the user exits the maneuvering area, see figure 3.

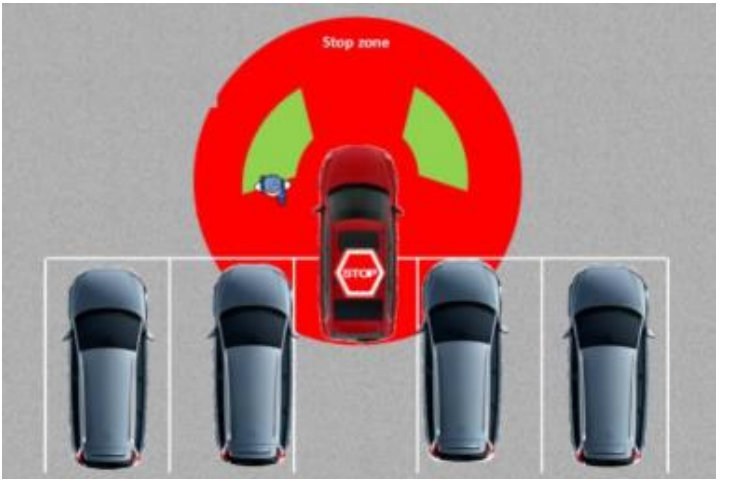

Figure 3. The car stops when the user exits the maneuvering area.

Park in

When Virtual Leash is used for Park in scenarios, the driver finds a parking spot, and puts the car in the starting position shown in figure 4.

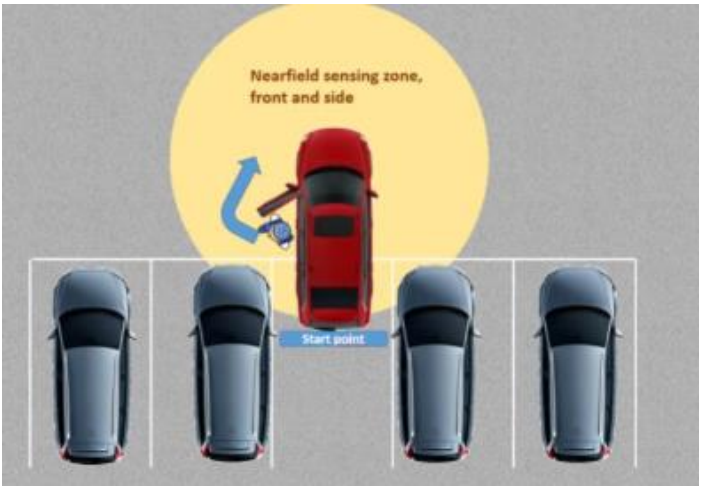

Figure 4. Starting point of Park in with Virtual Leash.

The user requests Park in with Virtual Leash on the dashboard in the car. Once it has been requested the car waits for the driver door to open. When it has been opened the Passive Entry system of the car scan for the key at the driver side of the car and then at the front of the car (or at the back, if the parking is done in the other direction). Once the key has been detected in the front, the ultrasonic sensors start to scan and the user has to complete the same initiation sequence as for park out, described in the Park out section above, see figure 5.

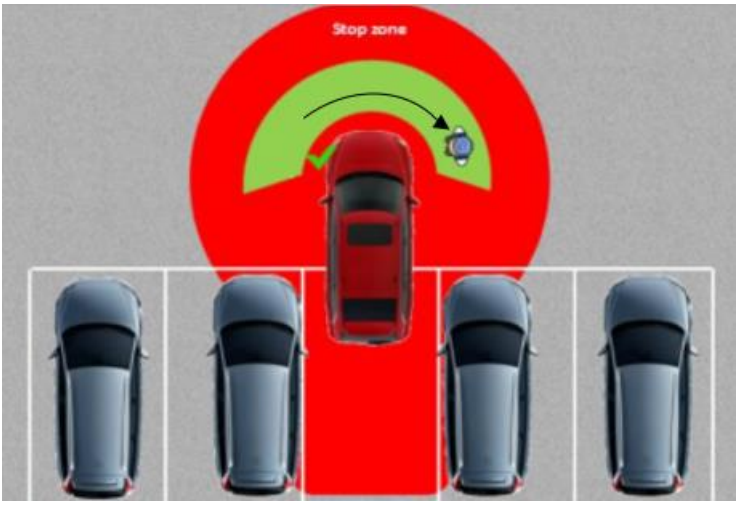

Figure 5. Initiation sequence completed for Park in.

After this, the user is allowed to lead the car in to the parking spot. The Virtual Leash ends when the parking maneuver is completed or if the driver walks out of the maneuvering zone shown in figure 6.

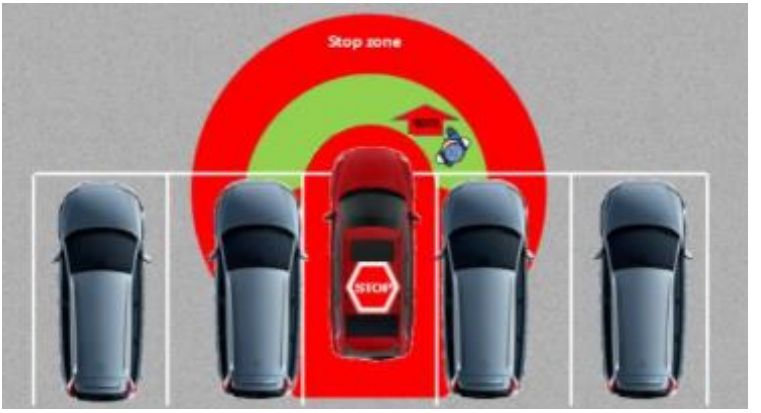

Figure 6. The car stops when the user walks out of the maneuvering zone.

1.2 Problem statement

The purpose for this thesis work was to create a proof of concept for the Virtual Leash function using ultrasonic sensors. This was going to be done by creating a simulation model in Simulink that could simulate and test the Virtual Leash

function. The goal was to have a model that could run recorded logs from a test vehicle, and also to be connected to the CAN bus in a test car and run in real time. The scope of the thesis work was limited to only handle the ultrasonic data, and therefore all other signals that would be used for the real case was mocked in the model. The propulsion of the car would not be implemented, instead the car would be driven and the model would evaluate if the car moved correctly.

2. Ultrasonic Sensors

2.1 Ultrasonic sensor

An ultrasonic sensor is a device that emits and receives a soundwave with a frequency higher than can be detected by the human ear, around 20 kHz. By measuring the time it takes for the soundwave to come back to the sensor, one can calculate the distance to the object that the soundwave was reflected on. This is calculated by $r = c * \Delta t/2$, where r is the distance, c is the speed of sound, and t is the time. An approximate relation of the speed of sound can be derived from the ideal gas law, where $c \approx 331{,}4 * \sqrt{(1 + T/273{,}15)}$ m/s. However, the speed of sound is temperature dependent and at 0°C the it changes by 0,6 m/s/ °C and from -30°C to 40°C it varies from ca 312 to 355 m/s. This needs to be takes into account and most ultrasonic sensors on the market compensates for this.

Other factors that also will have an impact on the speed of sound is the humidity of the air and the air pressure. However, compared to the air temperature these are have a very small impact and are sometimes neglected. [1, 2, 3]

2.2 Sensor setup

For this project a new generation of ultrasonic sensors were used. These sensors transmit short ultrasonic impulses with a varying frequency of 48-57 kHz that are reflected by the surrounding objects. The processing of the returned signal is done in a standalone ECU from the supplier and sent to the user PC through a CAN driver from Vector, more on this in chapter 5.3.

There are six sensors in the front of the car and six in the back. The sensor positioning is shown in figure 7.

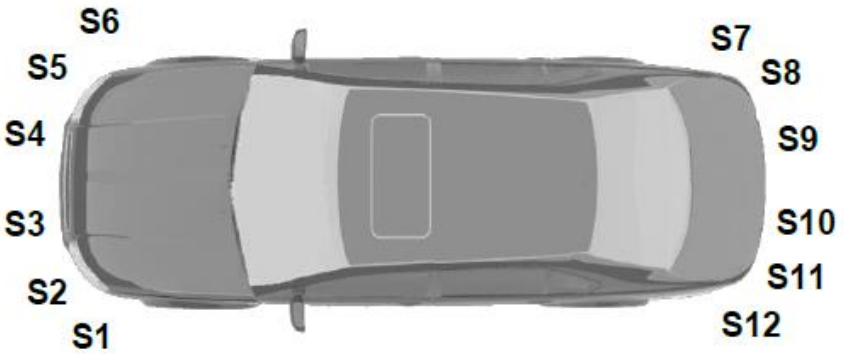

Figure 7. The positioning of the ultrasonic sensors.

Every sensor returns one direct echo, meaning it registers the echo that it transmitted, and two cross echoes, which means that it registers an echo from the sensor positioned next to it. This means that sensor four, marked S4 in figure 7, can register one direct echo that it transmitted itself and two cross echoes, one from S3 and one from S5. The cross echoes are primarily used to triangulate and evaluate the accuracy of the objects detected, but they can also be used as another echo signal.

The sensors are able to detect object at a distance from 15 cm to 510 cm with an accuracy of 3-15 cm.

3. Method

Figure 8 describes the flow of events in the Virtual Leash function.

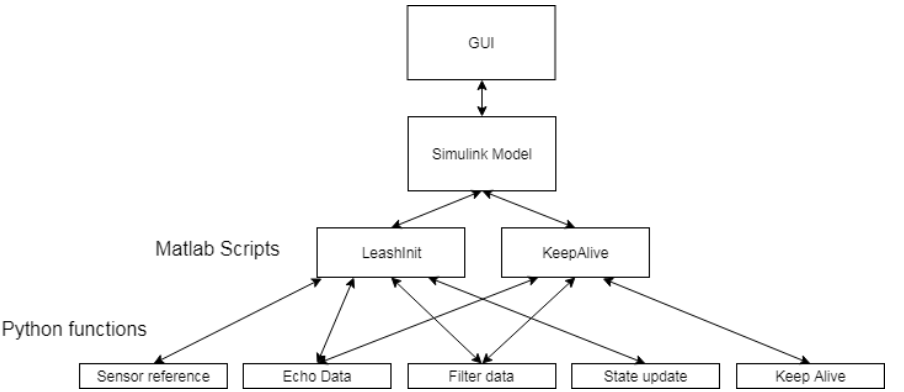

Figure 8. Communication between the different parts of the model.

The simulation was set up with a state flow model in Simulink that was started from a GUI created in Appdesigner. The state flow model changed states from park in to park out, park forwards or backwards etc., depending on the input data from the GUI.

When all required conditions for the start of the Virtual Leash had been met, the state flow model called a Matlab function, called LeashInit, which started the initiation sequence of the Virtual Leash function. The Matlab function called a Python script that read ultrasonic data from a log file or from the CAN bus in a test car in real time. The Python script then filtered the data and changed states in the initiation sequence.

When the initiation sequence is finished, the Matlab script returns that information to the state flow model which puts itself in the Keep Alive state. This state calls another Matlab function, called KeepAlive, which calls another Python script that reads the ultrasonic data and handles the Keep Alive scenario.

The information regarding which state the initiation sequence is in, if the user is detected, if the initiation sequence is completed, and the status of the Keep Alive is continuously fed back to the GUI so the user can track the progress.

3.1 Simulink model

The Simulink model consists of two state flow blocks and a number of input variables, see figure 9.

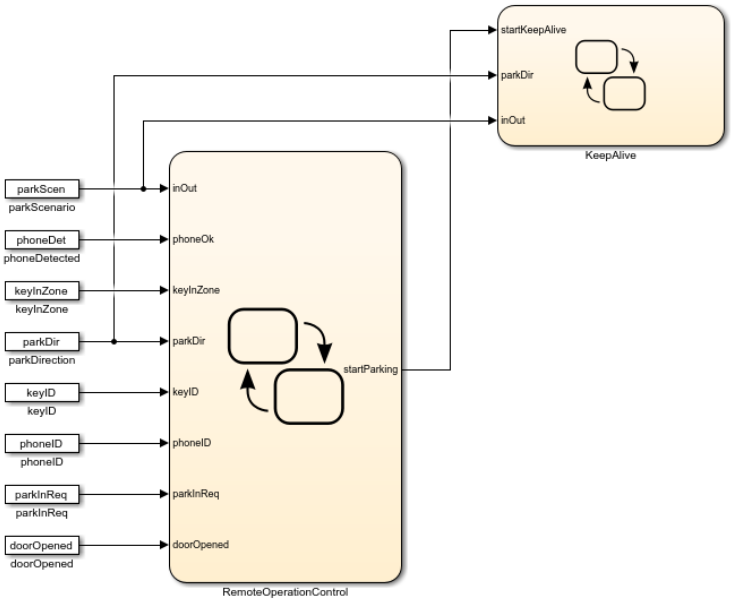

Figure 9. The State flow model in Simulink.

The inputs to the state flow blocks comes from a GUI shown in figure 10 below. The GUI can start, stop and pause the model, as well as changing the models input variables by flipping the switches in the model

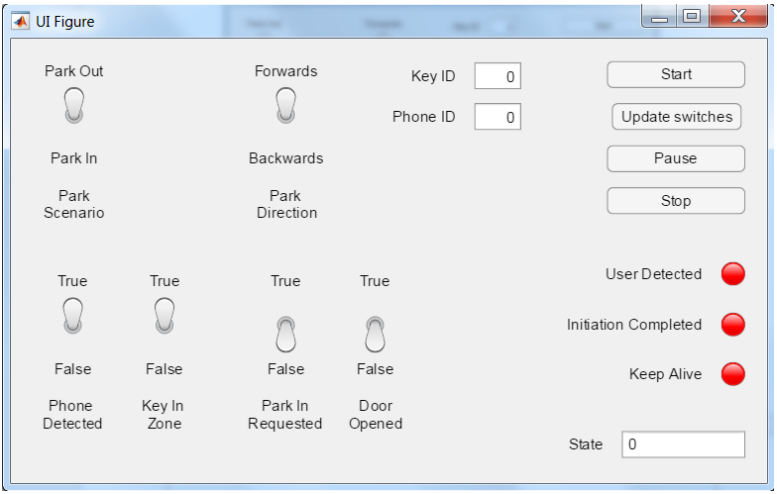

Figure 9. The GUI that controls the model.

Remote Operation Control

The RemoteOperationControl block handles the initiation sequence of Virtual Leash. The first thing that the RemoteOperationControl block does is check whether the user has requested park in or park out, see figure 11 below. Depending on the variable inOut the RemoteOperationControl-block enters either the ParkIn-block or the ParkOut-block. The simulation continuously checks that value of the inOut variable and if it changes during the simulation, if also changes states between ParkIn and ParkOut.

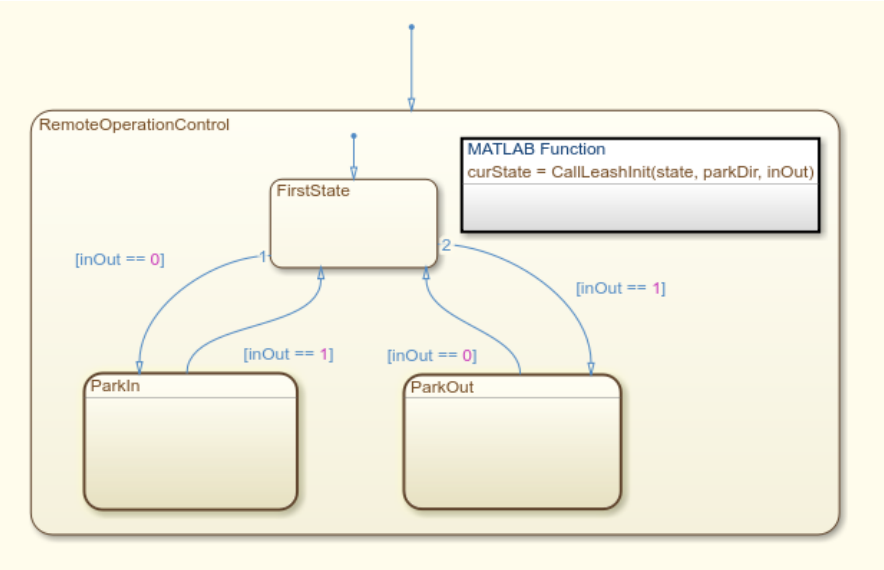

Figure 11. The RemoteOperationControl block chooses park in or park out depending on the input data.

Park in

The ParkIn-block handles the park in scenario of the initiation, see figure 12. When it enters the ParkIn-block it enters the InCar-state and sets the variable startParking to false. startParking is the variable that starts the KeepAlive block once the initiation is completed. The parkIn block first waits for the user to request park in in the GUI. When the user does, the parkInReq variable turns to 1. After this the block waits until the signals for doorOpened and keyInZone are true. These signals represents that the driver door opens during a park in and that the key paired with the users phone is recognized by the Passive Entry system in the car. In the simulation these signals are mocked by the GUI.

Once the doorOpened and keyInZone signals are both 1 the ParkIn-block enters the OutOfCar-state. It then looks if the parking is going to be done forwards or backwards, which is chosed by the user in the GUI, and depending on that it enters ParkInFront of ParkInBack. If the keyInZone and doorOpened variables turns to 0 during the simuation the model will return to the InCar state.

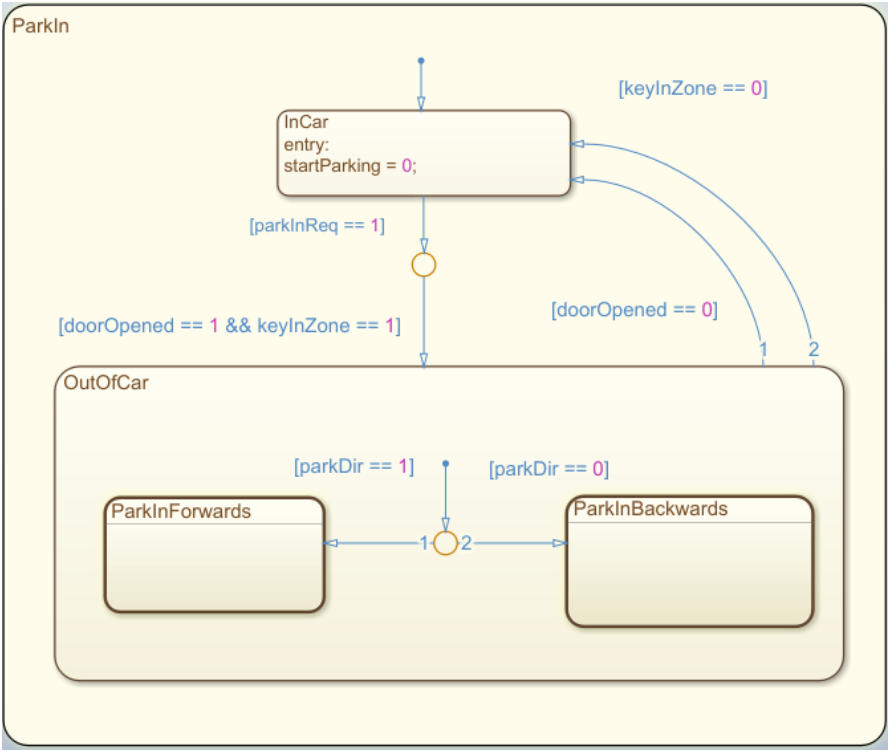

Figure 12. ParkIn block.

The ParkInForwards- and ParkInBackwards-blocks contain the same things. As seen in Figure 13, the ParkInForwards-block calls LeashInit-function. LeashInit is a Matlab function that handles the initiation sequence of the Virtual Leash. The function inputs are the current state, parking direction (forwards or backwards), and the parking scenario (park in or park out). The function returns the new state of the initiation sequence. If the function returns a state of 7 or -7 the model will enter the InitiationCompleted state, which sets the startParking variable to 1, which starts the KeepAlive block. If the returned state is not 7 or -7 the GetState state calls the LeashInit function again, as seen in the figure 13.

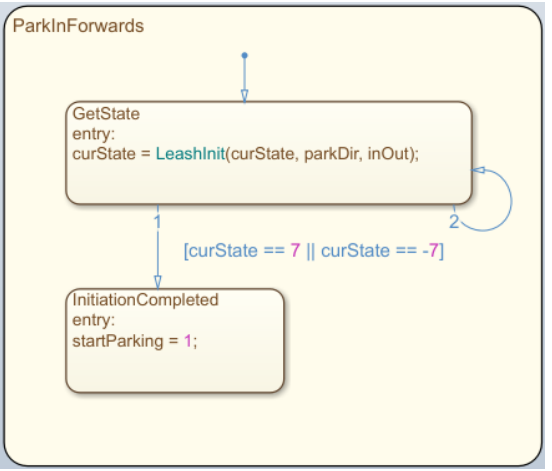

Figure 13. ParkInForwards-block.

Park Out

The same way as the ParkIn block handles the park in scenario of the initiation sequence of Virtual Leash, the ParkOut block handles the park out scenario, as seen in figure 14 below. When it enters the ParkOut block it sets the startParking variable to 0. It then waits for the phoneOk variable to turn to 1. The phoneOk variable represents that the park out sequence has been requested by the user and that the phone is detected by the car via Bluetooth. These conditions are mocked by the GUI in the simulation.

When phoneOk is 1 the model checks if the keyID and the phoneID variables are the same. These variables are put into the GUI, and they represents that the key that the car is detecting is the same key that is paired with the phone that is detected through the Bluetooth.

Once the conditions described above are true the ParkOut-block enters the TrackUser-state. Here it will enter ParkOutForwads or ParkOutBackwards depending on the value or the parkDir variable, which represents parking direction.

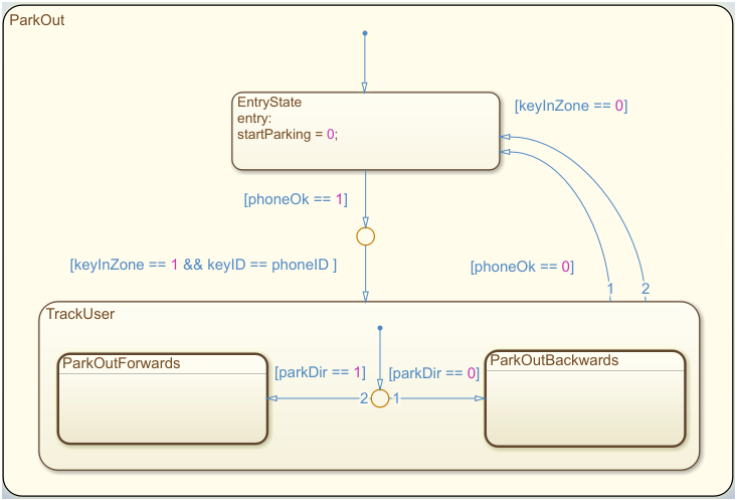

Figure 14. ParkOut block.

The ParkOutForwards- and ParkOutBackwards-blocks work the same as the ParkInForwards-block, see Figure 13.

Keep Alive

The KeepAlive-block starts in the WaitingToStart-state, as seen in figure 15. Once the RemoteOperationControl sets the value of the startKeepAlive variable to 1, meaning the initiation sequence is completed, the KeepAlive block will enter the CallScript-state and call a Matlab function called keepAlive. The inputs to the function are parking direction, parking scenario and keep alive state, which represents if the car is allowed to follow the driver or not. The outputs from the function is the new keep alive state.

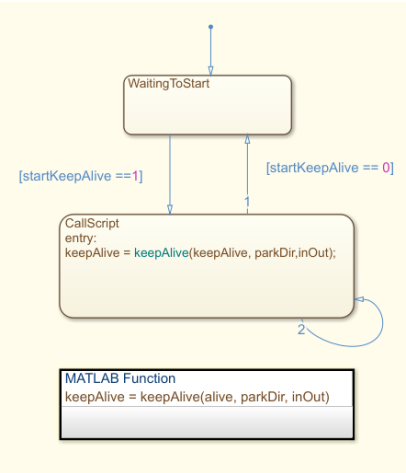

Figure 105. KeepAlive block.

3.2 Matlab functions and Python Script

LeashInit

LeashInit is the Matlab function that handles the initiation sequence of Virtual Leash. When it enters the function it imports the saved data from the Model Workspace in Simulink. The data consists of the previous echoes, timestamp and state from the previous function call.

When it has imported the data it calls a Python script that receives the current echo data from a log file or from the CAN bus in the test car, depending on if the user is running the model in simulation mode or in real time. The Python script will call a method that creates an app for CANoe, which is a software used to read CAN data. The Python script then reads the echo data from CANoe and returns it to Matlab. Once it has the new echo data it filters it according to the formula below.

$$filtered\ echo = old\ echo * 0{,}6 + new\ echo * 0{,}4$$

The script also sets a maximum value of the echo data at 300 cm. This is done because in the initiation and the keep alive sequences in the script never look at values greater than 200 cm.

When the filtered data is passed back to the Matlab function it calls another Python method that evaluates which state the initiation sequence is in. Depending on the state, the method looks at the next position in the initiation sequence. There is a timer that resets the initiation sequence if the user does not reach the next state

within one second. When the user has completed the initiation sequence the script returns that information to the state flow diagram in Simulink and that triggers the Keep Alive block to start.

KeepAlive

KeepAlive is the Matlab function that handles the part of the Virtual Leash when the car is starting to move. When the initiation sequence is completed, the KeepAlive function starts. KeepAlive imports the saved data from the Model Workspace in Simulink. The data consists of the previous echoes, the position of the user, and state from the previous function call.

KeepAlive calls two Python scripts that reads and filters the echo data from CANoe in the same way as LeashInit above. It then calls another Python script that evaluates whether or not the user is in the right position to operate the Virtual Leash. The script looks at where the user is supposed to be positioned and compares that to the echo data from the sensors. If the user is not in the right position the Keep Alive lamp in the GUI turns red, which represents that the car would have stopped in the real case.

During a park out sequence the driver has to be positioned at the end position of the initiation sequence, between 50 and 200 cm from the car, at the cars side, as seen in the figure 16.

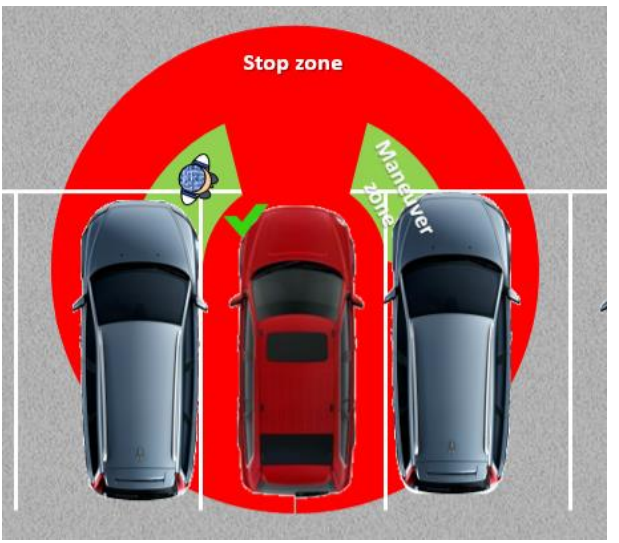

Figure 16. Maneuver zones for Virtual Leash during park out.

If the user steps out of the maneuver zone the KeepAlive script will pause the Virtual Leash sequence and wait for the driver to enter the maneuver zone again. If this takes too long the script will exit the sequence.

A park in sequence is performed in the same way as a park out. However, the maneuver zone is not limited to the sides of the car since the car is not moving towards you, as seen in figure 17.

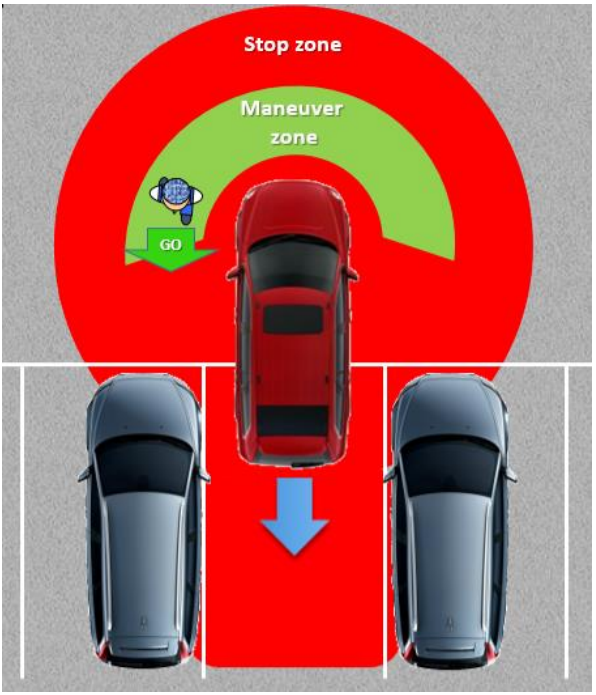

Figure 17.Maneuver zone for Virtual Leash during park in.

3.3 Car setup

The setup in the car is shown in figure 18.

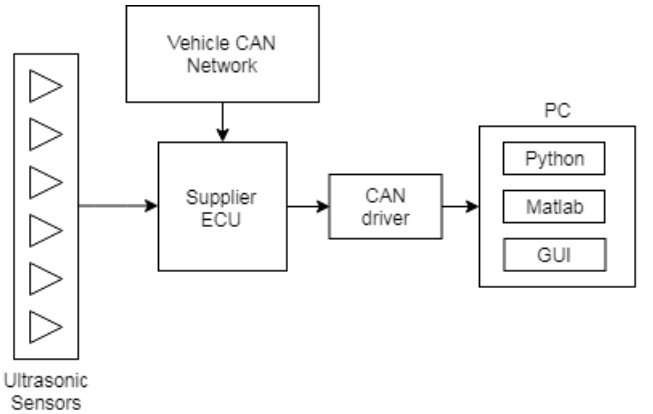

Figure 118. Figure showing the setup for the ultrasonic sensors in the car.

The ultrasonic sensors are connected to a standalone ECU from the supplier. The ECU is connected to the cars Chassis CAN network. The ECU sends the ultrasonic data to the PC that we are running the software on through a CANFD protocol. Due to the fact that the PC cannot read a CANFD protocol we have used a Vector CAN driver to read the data. The data was then read by the PC through CANoe, which is a CAN processing software from Vector.

4. Results

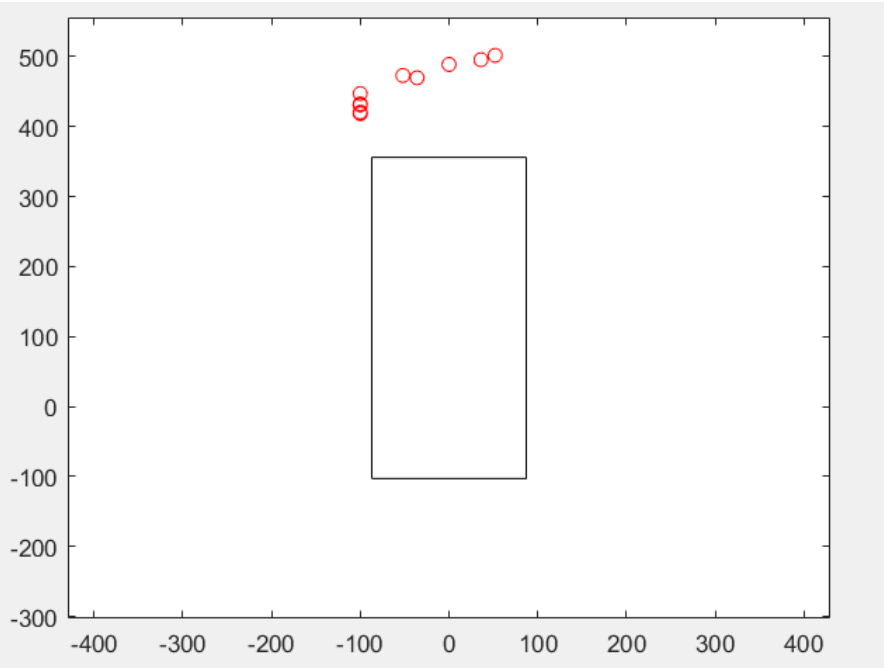

Figure 19. Plot of the user position during the initiation sequence and the park out.

Figure 19 shows a plot of the position of the user during the initiation and Keep Alive sequences. The plotted position is relative to the car, which means that when the car is being led out of the parking spot the plotted position in the graph does not change much because the car itself is also moving.

The position of the user is plotted every time the user enters a new state in the initiation sequence and every fifth loop in the Keep Alive sequence.

The progress of the Virtual Leash can be tracked in the graph, but also in the GUI, see figure 20. The three green lamps at the right end of the GUI turn green when a user is detected, when the initiation sequence is complete, and when the car is allowed to follow the driver. Otherwise they are showing a red light. The Keep Alive lamp turns green when the initiation sequence is complete and when the user is located within the maneuver zone. The State field at the right bottom corner of the GUI shows what state the initiation sequence is in. This states go from 0 to 7 if the initiation is done from the right side of the car to the left side, and from 0 to -7

if it is done from the left side to the right. The states represents how far through the initiation sequence the user has come. State 7 is entered when the Keep Alive script stars running.

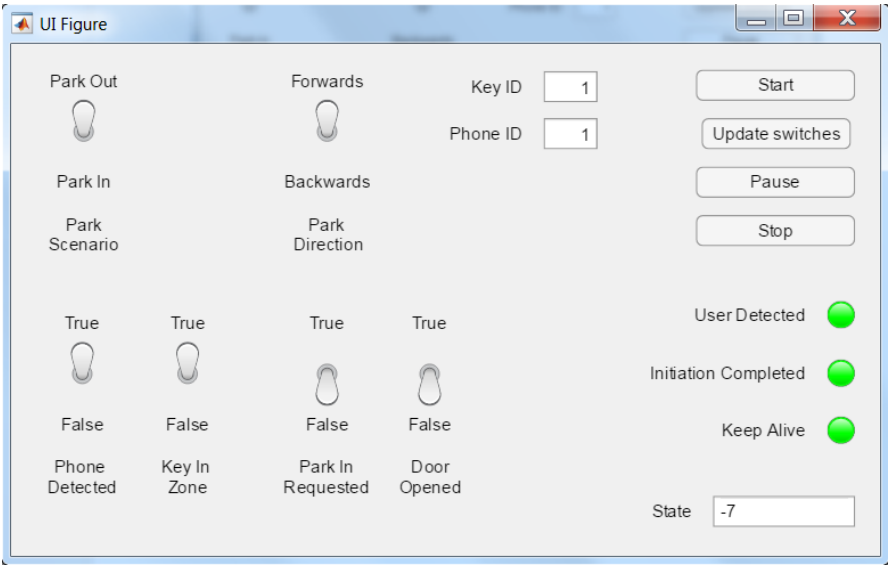

Figure 20. GUI showing the progress of the Virtual Leash.

In figure 21 below we can see a use case where the user did not complete the initiation sequence and therefore only four positions were plotted.

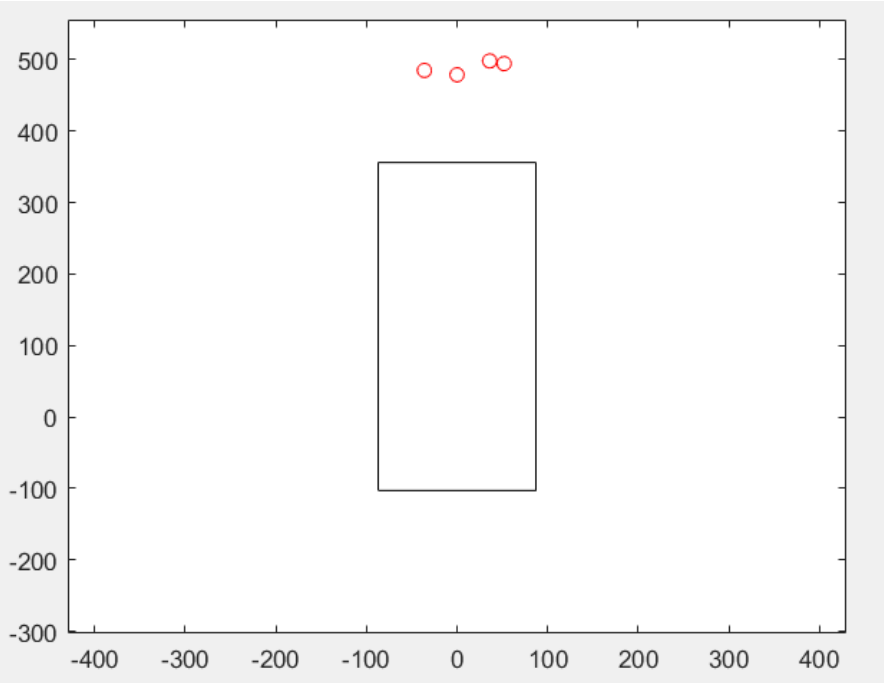

Figure 21. Graph showing an uncompleted initiation sequence.

Figure 22 below shows the GUI for the same use case. The lamps for the initiation and keep alive sequences are red and the State field in the bottom shows that the user never made it passed state -5.

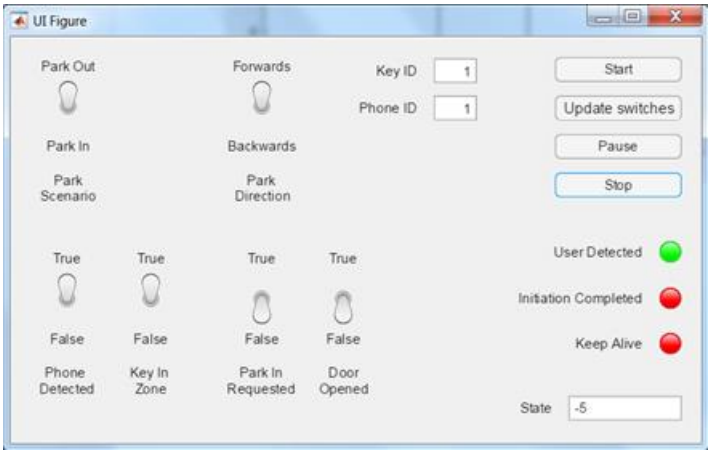

Figure 22. GUI during an uncompleted initiation sequence.

When the car is moving it is important to make sure that no other persons or objects are in front of the car. This was implemented for the Keep Alive function, but not for the initiation sequence. The reason for this is that the ultrasonic sensors were not able to give us a useful object list. Therefore we had to rely on the echo data from the sensors, and because of the wide spreading angle and the poor quality of the echo data we were not able to distinguish different objects from each other with an acceptable accuracy during the initiation sequence.

5. Discussion

The purpose of this thesis work was to evaluate if ultrasonic sensors could be used for the Virtual Leash function. Even though all aspects of the function were not able to be implemented, the overall results look promising.

As mentioned in the previous chapter, there were some performance issues with the ultrasonic sensors. Because of the fact that the sensors could not provide object lists with object IDs and position connected to each object as planned, the model had to rely only on echo data from the sensors. This meant the model had to scan for objects, and once an object was found the model had to anticipate where that object was going. With a reliable object list the model would be able to find an object and know where it is located and where it is moving. This would make the function more reliable and it would probably also make it easier to implement as well. However, the echo data that was used is good enough to show that the function could be implemented using ultrasonic sensors.

The fact that the model was not able to distinguish different object from each other also was an issue. If another person or object gets in front of the car during the initiation sequence the model is supposed to stop and wait for the user to restart the initiation sequence. This is a safety requirement for Virtual Leash. Due to the fact that the sensors could not provide an object list, the model had to try to distinguish different object form each other using the echo data. However, because of the wide spreading angle of the sensors the user could be detected by up to four different sensors at the same time. Therefore, the model was not able to know if the sensors the user had passed had picked up another object or if they were still detecting the user. As a result, this part of the function could not be implemented.

The conclusion that can be drawn from this thesis work is that ultrasonic sensors could be used for the Virtual Leash function. As already mentioned, to be able to develop a reliable function the sensor performance needs to be improved. If the supplier can solve these performance issues, it is possible to think the function can be implemented with the desired reliability. Instead of anticipating where an object is going the model would be able to find an object, get its position relative to the car and see where it is moving. For safety and reliability this is a huge difference.

There are some things that could be interesting to develop further in the future. Once the performance issue is solved the whole function should be implemented using the object list. As mentioned above, this would increase reliability of the function. It would also be very interesting to be able to test the Passive Entry system

in the car. The Passive Entry system was not accessible during the thesis work, and it was also considered out of scope, so these signals were mocked in the model. It would be very interesting to try to match the Passive Entry signal from the key to an object from the ultrasonic sensors to confirm that object as the user.

Another interesting area is sensor fusion. It would be very interesting to use the ultrasonic data together with camera data. This could help the model to understand what kind of object the car is detecting and differentiate people from other cars or static objects, which could make the function better in several ways. It would make it easier to detect the driver, increase the reliability of the function, and estimate the safety risk in different situations depending of what kind of object is getting closer to the car.

To summarize, it would be fair to say that the thesis work shows that ultrasonic data could be used to implement the Virtual Leash function. The fact that most of the function was implemented successfully while only relying on the echo data is very promising. If the object lists from the sensors would be accessible it would make the function more reliable, it would make it possible to implement the whole function, and it would probably make the function easier to implement.

6. References

[1] Volvo Car Corporation. 2016. *Virtual Leash – Device Less Remote Auto Parkin*g. Gothenburg, Sweden.
[2] Blitz, Jack. 1967. *Fundamentals of Ultrasonics*. 2 Ed. Butterworths, London.
[3] Lindstedt, Gunnar. 1996. *Borrowing the Bat's Ear for Automation – Ultrasonic Measurements in an Industrial Environment*. Lund, Sweden.
[4] Nordevall, Johan. 2015. *Method Development of Automotive Ultrasound Simulations*. Chalmers University of Technology. Gothenburg, Sweden.