Inducing trust in an Autonomous Vehicle using a socially interactive agent: LISA

Archita Gorle
1037283
a.gorle@student.tue.nl

Arun Tom Skariah
1033127
a.t.skariah@student.tue.nl

Bas Bakx
0806552
b.c.j.bakx@student.tue.nl

Geert Jaspers
0823217
g.j.jaspers@student.tue.nl

Sietze van de Star
1306575
s.v.d.star@student.tue.nl

Sri Harshini Sri Ramulu
1033968
s.h.sri.ramulu@student.tue.nl

ABSTRACT

Until recently, the idea of a driverless car seemed to be limited to a concept for science fiction. But this scene has been changed to a great extent in just a few short years as the technology behind autonomous vehicles has taken huge strides. However, many challenges remain within the area of human factors, such as user's trust for Autonomous Driving (AD). This paper aims to address the issue of trust in two contexts, the first is by investigating how hand-over and take-over procedures can be handled in a way that is clearly understandable for both the human and the machine by the implementation of an intelligent agent. The second context aims to induce trust by social interaction through gamification. To facilitate this, an intelligent and interactive agent called Lisa is created. To support the functionality of the first part of this project, a supervised machine learning technique, particularly Support Vector Machine learning algorithm is used to monitor the behavior of the driver. The testing data set of the SVM has yielded an accuracy of 94.65%. This implies that the agent can very accurately detect the driver's fatigue. The game has been developed using Google cloud API for speech recognition, which works with an accuracy of 80% in detecting the user's voice. The interaction starts when the agent is invoked by it's name, LISA.

1. INTRODUCTION

1.1 Vision

Today, as the transition from level 2 to level 3 automation is happening slowly, we can say that the pervasion of autonomous cars is gradually increasing, very soon fully autonomous driverless cars will be a reality. According to Tesla [1], autonomous driving is bound to increase confidence in the person behind the wheel, and make driving more enjoyable. As the transition from level 2 to level 3 automation is happening, these early Autonomous Vehicle (AV) experiences should be considered as trust-building opportunities among the car and the users. Before the drivers start to completely take their hands off the wheel for the first time, the AV should gain rational and emotional trust from the users [2]. To accomplish this, the driver needs to be able to see and interpret the information the car is using to execute its actions. This project aims to address the issue of trust in an AV by establishing an effective interaction between the user and the AV. The AV on the whole can be viewed as an intelligent agent that is able to convey certain messages to assure the driver to perform tasks or inform the driver about the tasks performed by the AV. The important question here is: how does the AV convey this information to the driver?

As the features of cognition are deeply dependent on the characteristics of the physical body or beyond-the-brain body of an agent [2], the agent can be embodied to take a certain form to communicate with the people in the car. Currently, an embodied agent using intelligent algorithms in AVs can be used to let the driver know when to take over control or when to hand over control and to entertain people in the car when it is in a fully autonomous mode. Above all, embodied intelligent agents are used to gain the trust of the driver during the initial stages of automation, by showcasing characteristics of situational awareness and also establishing a form of interaction between the agent and the people in the car. This is to ensure that there is a smooth transition to an era of fully autonomous cars, where there is an effective rapport and social interaction among the users and the AV.

1.2 Problem Statement

With the emergence of Autonomous vehicle technologies, many challenges remain within the area of human factors, such as user trust for Autonomous Driving (AD). This paper aims to address the issue of trust in an autonomous vehicle in level 2 automation, through the use of social cues. This will be done in two ways:

1. The first way is to investigate how hand-over and take-over procedures can be handled in a way that is clearly understandable for both the human and the machine by the implementation of an intelligent agent.
2. The second way is to understand how an appropriate level of user trust for AD vehicle systems can be created via human-machine interaction (HMI). This problem is addressed by implementing an intelligent agent that could aid social interaction among passengers as well as interaction of the user with the agent through gamification facilitated by speech recognition.

1.3 Related work

An important characteristic of an AV in the level 2 of automation is that the vehicle can drive relatively independently for some time, but it can request the human driver to assist or take over control through an alert. When a highly automated car reaches its operational limits, it needs to provide a take-over request, when AV wants the driver to take over control in situations where it needs driver intervention [9].This raises the question of how to alert the driver? Heiden, Iqbal and Janssen [3] present a research on handover control in semi-autonomous vehicles.

Handover requests require the drivers to take control instantaneously, they suggest that auditory pre-alerts before the handover request impact the success of handovers. They performed a study with a driving simulator, where the drivers received a repeater burst audio pre-alerts before the handover. Their results showed that the drivers focused more on the road before handover occurred, disengaging from previous tasks, resulting in a safe handover.

Also, [9] proposes a generic process for takeover from a full self-driving vehicle to manual driving. This proposal tells us that the handover involves a 'take-over-request' (TOR) from the system which will then followed by takeover by the driver. Further, the driver behavior on takeover is investigated by conducting experimental tests on participants, giving them different situations of hazards. The study ended on a note saying that drivers felt comfortable during the takeover process when alerted using multimodal alert signal but further research should be done in the area of car-driver handovers. [11] also insists that a multimodal language-based warning systems yield better results than unimodal warning systems. Similar study is also performed by [12].

While the feasibility of takeover has been discussed, the literature in [10] discusses on the take-over time, focusing on which point in time should the driver be alerted and directed back to the driving task. The paper considers two take-over times and studies the effects of automation in both periods. The results indicate that with shorter take-over time, the decisions are quicker but worse. The study also tells that the main-study subjects performed poorly compared to manual drivers by using more brakes and unnecessary accelerations, suggesting further investigation on safe take-over times.

In a paper on trustworthy automation, Rezvani et al. [4] present three kinds of user interfaces to assist hand over by considering the effects of expression of internal and external awareness. Internal awareness is knowing about whether the system is confident in its ability to handle the current situation, whereas external awareness is being able to identify the limitations of the system, as the car is driving, in terms of situational anomalies. They conducted an user study on the different levels of information that has to be presented to a driver and the effects of expressing different levels of information about situational awareness and trust in automation. Their results showed that effectively displaying information about the external awareness on different anomalies on the road, the driver's situational awareness increased which in turn increased trust in the system and performance after the transfer of control.

Another issue is overtrust in highly automated cars [13]. In June 2016, a man was killed in his self-driving Tesla Model S because a crossing truck was not detected as an obstacle [14]. The driver did not react because the car made him feel safe and he stopped paying attention to the road. The Autopilot function in the Tesla Model S requires the drivers to keep their hands on the wheel and will come to a halt if they do not do so[15]. He overtrusted [13] the car, kept his hands on the wheel but did not pay attention and therefore he could not prevent the accident by taking control of the vehicle back. In order to avoid such incidents, it is essential that the car detects the driver's fatigue and alerts the driver for quick take-over or handover procedure. This demands for a more clearer interaction between the car and the driver.

Finally, research presented in [16] shows that handovers are feasible, however, they are not a satisfactory solution since human factor issues such as reduced situation awareness arise in automated driving. it is suggested to implement cooperative interfaces to enable automated driving even with imperfect automation. The research also recommends to consider four basic requirements for driver–vehicle cooperation: mutual predictability, directability, shared situation representation, and calibrated trust in automation.

2. BACKGROUND INFORMATION

This section consists of information about the theories, principles and learning algorithms that are used in the design solution.

2.1 Design approach

To address the problem of trust in an Autonomous Vehicle, a technology oriented approach is chosen. The aim is to create an agent, LISA who is interactive and intelligent to detect the behavior of the driver. As mentioned in section 1.2, the first part of this project aims to produce a design solutions for knowing how and when the agent should request the user to hand over control and or even take over the control of the user in case of a possible dangerous situation.

The intention of the system is communicated to the user with the help of an agent which keeps track of the driver's behaviour, in case the system is confident that the driver is distracted the agent will ask the driver to hand over control. When the driver is still distracted and does not respond to the warning signs it lets the user know that it has taken over the control of the vehicle. The agent conveys these messages at different levels (1 to 5) of certainty (see Fig.1). The levels are chosen according to Donmez, Boyle and Lee [5] who use a 3 second moving average measurement to assess distraction. On top of that a two second window was chosen for the driver to respond to the warning: either he does not respond and the system takes control, or he agrees with the request and hands over control or he starts focussing on the driving task.

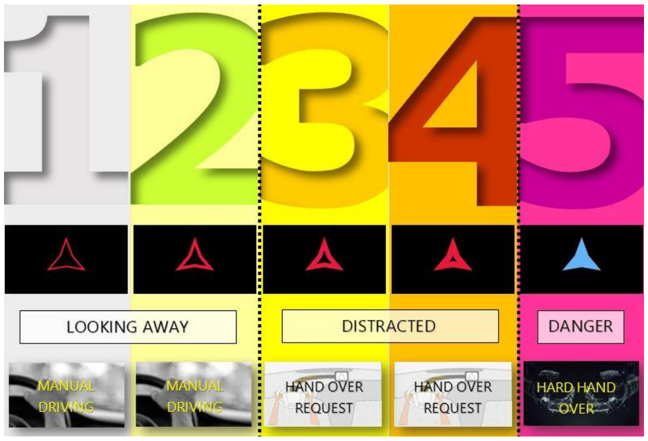


Figure 1. Different levels of certainty of the agent

The second part of this paper, tries to create trust on the Agent by creating situational awareness and to enhance social interaction through gamification. For this, a speech based game is used because a speech based interaction with in an in-Vehicle agent can be used to increase driver's performance [5] and also be used as a tool to create trust on the vehicle. The game that is used in the paper is Ispy [6], a guessing game where one of the players playing the game is a spy. The other players guess what the spy is thinking. This can be used to create situational awareness, when the agent spies at something that it sees in its surroundings. Not only does this create situational awareness it also helps the passengers develop an affinity towards the agent by creating a social interaction. Moreover, the interaction starts only when the user invokes the agent by calling her name, Lisa.

2.2 Primary social behaviors

Social cues can play an important role in aiding understanding between the driver and the vehicle. The functionalities of an autonomous or highly autonomous car would be very complex involving various internal states and the driver may not be able to see a current state due to the complexity. This might affect the trust of the driver about the controls of the vehicle. To address this challenge, a smart agent (avatar) is introduced with embodied intelligent behaviour and social cues. The social cues we included are:

- Eye contact
- Conversation
- Reading mental state
- Persuasion
- Interpersonal Interaction

Social cues help in expressing the intentions during social interactions. The most important ones being eye contact, facial expressions, body language, vocal tone and body posture. These social cues are also important in building trust and confidence among the people involved in an interaction. Research [19] shows that mirroring body language is a way to bond and to build understanding and that people who experience the same emotions are likely to experience mutual trust, connection and understanding. Eye contact is a form of body language which is important during communication. The old saying that "eyes are a reflection of your inner self" holds true in most cases and this formed the base for using eye gaze detection of the driver by the agent to understand his state of mind, thus estimating if handover and takeover actions are needed.

Semi-autonomous features have proven to improve vehicle safety so long as drivers continue to pay attention when vehicles or other objects suddenly enter their path. Mentioning the Tesla accident [13] , it could have been avoided if the driver had paid attention to the system's state. The driver can be made to be attentive by using persuasive technique by constantly monitoring the driver's behaviour and giving handover requests.

People trust other people more if they act similar to themselves, because they compare their own actions with actions of others. Therefore social cues could be a way to abstract and visualize the complex functionality of these cars and create trust in such systems.

2.3 Learning algorithms

This project contains two parts. For the first part of the project, in order to track the behavior of the user using face recognition, a Support Vector Machine algorithm is used. Firstly, a Support vector machine (SVM), introduced by Vapnik [17] is a supervised machine learning technique used for pattern classification and for deducing relationship among variables. The underlying idea of SVM is to map training data into a higher dimensional feature space where it can be linearly separated.

Figure 2 and figure 3 shows the basic idea of classification using SVM. Consider labelled binary class training data denoted by D={(x_i,y_i)}, where x_i is a vector with multiple features and y_i is a class indicator with the values 0 or 1, as indicated by circles and stars respectively.

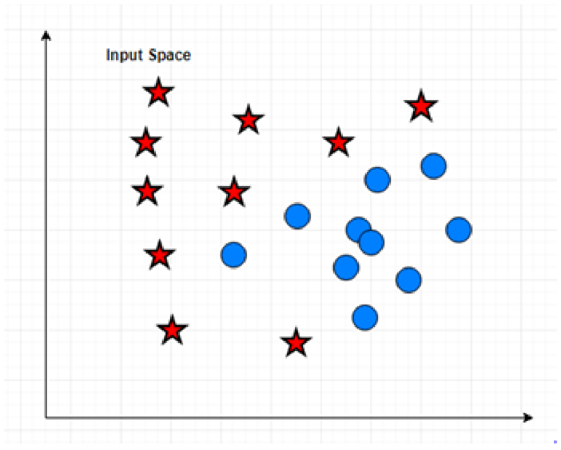


Figure 2: Input space for SVM algorithm

The input space is mapped into higher dimensional feature space using a kernel function. If the data is linearly separable in the new feature space, a hyperplane separating the data is generated by maximising the margin between both the classes as seen in figure 3.

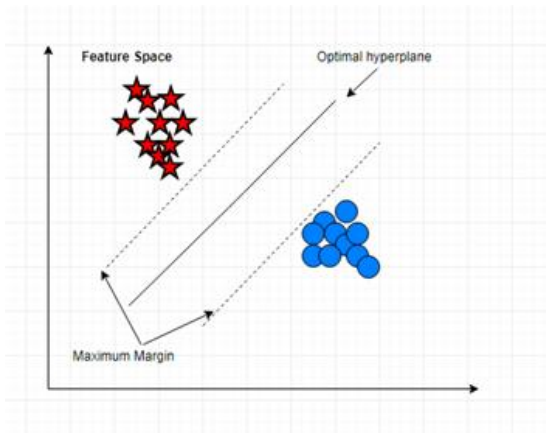


Figure 3: Hyperplane generated after data mapped to high-dimensional feature space

Although there exist multiple machine learning techniques, the learning technique of SVM makes it very suitable to be used in this project. Another algorithm that was considered was the Hidden Markov Model (HMM) [19]. Although, HMM algorithm considers transition from one state to another, it fails to consider momentary changes. Distraction is a momentary change that occurs suddenly while driving, for example if a mobile phone rings or a driver needs to take action to stop the kids at the back seat from quarreling. SVM can adapt better to these momentary changes. Moreover, SVM allows for the usage of multiple features, which can help to detect distraction more precisely. This makes it the best option for our project.

To establish a social interaction between the user and the agent using a game, we used a speech recognition API from the google cloud platform. Although no learning algorithm was used by us to make the game, the speech recognition API that is provided by Google uses deep neural networks to parse the Voice Input provided by the users. Deep neural networks are used to transcribe voice input to text, they are then compared to the data on the cloud platform.

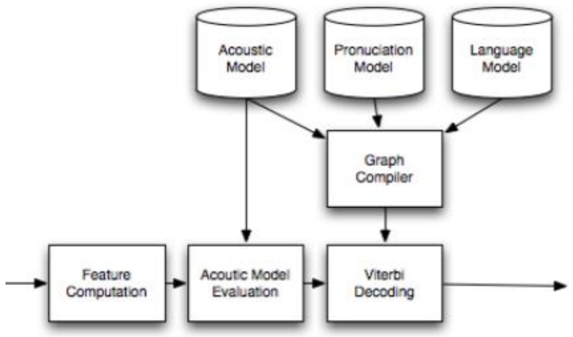


Figure 4. Basic block diagram of speech synthesizer used by google [9].

Once the voice string matches with the data on the cloud, the voice input is recognized. The API is available in python and is used to create a basic prototype of the game. The API requires to install a gTTs (google text-to-speech) module that can be used in python, this API provides a speech recognition module that works the same ways as the Google Voice Search works.

The speech synthesizer shown in fig.4 works with the accuracy of 80% [9] and is used in voice query applications on phone applications. The voice input is an acoustic signal which is parsed in three models [9]. Firstly, the acoustic model pays attention to the phonetic context of the speech input. Secondly, the pronunciation model enables the users to use different pronunciations of the same word uttered. Thirdly, the language model offers support for 80 different languages spoken by the users. All these models are mapped on to the Graph compiler which does the Viterbi decoding. Viterbi decoding uses a Hidden Markov model that converts speech to text. The acoustic signals are treated as a sequence of events and the string of text is the hidden cause of the acoustic signal, this algorithm finds the most likely string for any given acoustic signal. Thus, the voice input is converted into a string by the google voice synthesizer.

3. DESIGN AND SOLUTION

This section contains the details about the design scenario, the implementation and the results from the learning algorithm used and the testing and analysis of the prototype.

3.1 Design of the interaction - the social context

The scenario of this project is situated in an AV that is in a level 2 mode of automation, where the system is not fully autonomous. The major issue in this scenario is whether the driver trusts the system, this scenario boils down to one context, which is the focus of this paper. The driver may not be sure when to transfer control to the AV also the AV should be aware of the moment to take control or to transfer control to the driver. Thus, the AV is responsible for learning the behaviours of the driver, whether he/she is distracted, if that is the case the car should take over control.

A Microsoft Kinect 2 sensor is used to check whether a driver enters the vehicle. After the face detection algorithm has identified a face it checks whether the driver is focusing on the driving task or not. If the driver is not focussed the agent will turn red and starts slowly filling up every second the driver is not focussing. After 3 seconds of inattention the driver is no longer marked as merely looking away but as truly distracted. This will initiate a handover request by the system. The request is presented to the user by a walking movement of the lights of the LED strip, suggesting a requesting gesture. Finally, if the user after 5 seconds is still not responding to the signals, the system takes over control. At the final stage, the LED strip will fill up first followed by the appearance change of the agent, turning from red to blue.

Once the AV is in autonomous mode passengers are able to engage in a social interaction with the agent. The agent can be invoked by using it's name, Lisa. For this project the agent is able to play a game, I-spy, here the agent thinks of some object that it sees in its surroundings and the user takes a guess at what the agent looks at. The agent gives hints by mentioning the color and the starting letter of the object. For example "I spy with my little eye something blue and that starts with an S ". The game is used to enhance a social interaction between the agent and the users, thus building a rapport between the agent and the user, which leads to the users trusting the agent.

3.2 Intelligent behavior and embodiment:

The AV by itself can be considered as an embodied agent, but in order to represent the presence of the agent, it is embodied like shown in figure 5. The agent displays levels of certainty thus enabling the driver to understand whether there has to a transfer of control. Also, to create a social interaction with the agent, it is given a voice and can be invoked by the name 'LISA', which is built on the speech recognition API.

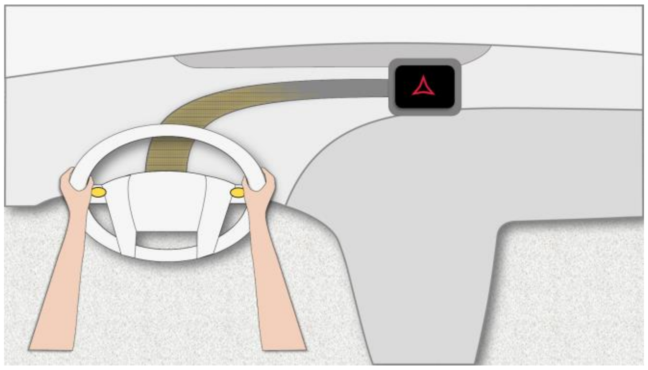


Figure 5. Positioning of the agent in the vehicle.

The first intelligent behaviour exhibited by the agent is, it can detect when the driver is distracted or alert. In order to create the model for the agent tracking the behavior of the driver an SVM is used. For creating a SVM test model, training data was collected. The data was collected using the Microsoft Kinect 2 sensor. Microsoft Kinect 2 comes with two API for face tracking: Face basics and Face HD. From this, Face HD API was used because it can be used to track face in 3D space which is incapable by Face basics.

The participants were given three tasks during data collection. The first task was to focus on driving whereas the other two tasks were to look left and right while driving. Each collected data point was labelled as 0 for being focused at task or 1 if the driver was looking away towards left or 2 if the driver was looking towards the centre console. The labels 1 and 2 was used to define driver being at the state of distracted from the primary task of driving.

The data collected was trained using linear SVM with 5-fold cross validation. The model produced an accuracy of 94.65%. Further details are mentioned in the confusion matrix (Table 1). Also the trained model is visually represented in figure 6.

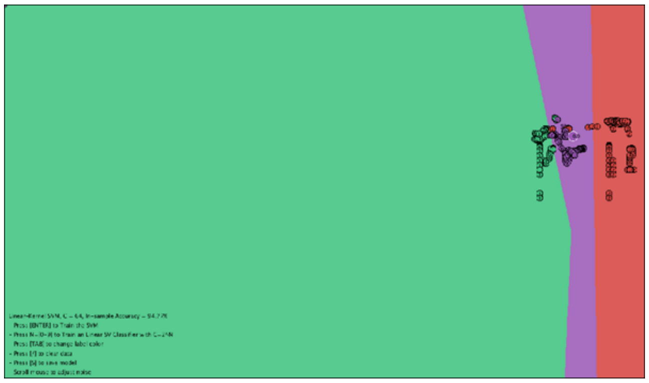

```
Linear-Kernel SVM, C = 64, In-sample Accuracy = 94.77%
  Press [ENTER] to Train the SVM
- Press N=[0-9] to Train an Linear SV Classifier with C=2^N
  Press [TAB] to change label color
- Press [/] to clear data
- Press [S] to save model
  Scroll mouse to adjust noise
```

Figure 6: Visual output after SVM training

	Predicted [0]	Predicted [1]	Predicted [2]	Total
Actual [0]	507	7	0	514
Actual [1]	66	392	0	458
Actual [2]	17	0	695	712
Total	590	399	695	

Table 1 : Confusion Matrix for the SVM model

Once the SVM model is generated, it is used to predict whether the driver is distracted or focused in real time. Depending on the state of the driver, it provides visual cues as mentioned in section 3.1. The code for training the SVM model and for real time implementation of the prototype is given in Appendix B.

The agent exhibits another intelligent behavior that plays a game with the user. For this, the gTTs and a speech recognition module is used, the speech recognition module recognizes the speech input given by the user. The code for the game is provided in appendix. C. The prototype of the game is successfully made, with the agent responding when it is invoked by its name. Although it works,it is not quite robust. It requires a strong internet connection and parsing longer voice input takes a longer time to be recognized.

3.3 Testing and analysis

Due to time constraints, usability testing was not performed. As mentioned in section 1.2 the first problem statement was to investigate how the hand-over and take-over procedures can be handled in a way that it is clearly understandable for both the human and the machine. The model presents an accuracy of 94.65% while predicting driver behavior, thus it is clear that the machine is able to interpret and understand the behavior of the user. But it is important to understand if the users are able to understand the behavior exhibited by the agent. What has to be known is that if the users find this behavior useful to perform the transfer of control. This can be done by testing the scenario in a driving simulator, by simulating potential hazards or situations that will result in the transfer of control. In the same way, the game can also be tested and a feedback from the users on how the social interaction helped create a rapport between the AV and the users could be measured from this.

4. CONCLUSION

Taking into consideration the challenges regarding trust and handover – takeover procedures, a smart agent that is socially interactive and intelligent is presented. The agent displays levels of certainty, enabling the driver to understand whether there has to a transfer of control. The agent also aids for social interaction in the car with the help of a simple game. The tracking of behavior of the driver is performed using face recognition, a Support Vector Machine algorithm while the speech recognition API that is provided by Google uses deep neural networks. The SVM algorithm was tested using a testing data set. With the agent being able to detect (focus or distraction) and read the state of mind of the driver at high percentages of predictability, the model facilitates for safer take-over and handover procedures.

5. DISCUSSION

As the transition from level 2 to level 3 automation is happening, today's cars are not fully autonomous. There is a certain level of trust that is required from the drivers on the AV, but over trust on the car is also not admissible. As discussed in section 1.3, the situation of over-trust on the AV which lead to a fatal accident because the driver was not focused on the road and the AV did not detect a truck as an obstacle. This makes it clear that both the AV and the driver should work hand-in-hand to avoid any such accidents. In this paper, we tried detecting the behavior of the driver, if he is distracted it alerts him to initiate a transfer of control. Although, it should be agreed upon that this paper only addresses a part of this problem by monitoring the behavior of the driver. Further research should be conducted in the area of car-driver handovers, in other situations (e.g. with the presence of other traffic participants, or more menacing circumstances) or real traffic. Beside the technical issues, liability issues also have to be taken into account, regarding the responsibility for accidents that occur in the transition phase from self-driving automation to manual driving. Car-driver handovers cued by a combination of auditory (i.e. earcons and speech) and visual (i.e. icons and text) warnings are a promising approach to compensate for situations wherein a fully self-driving vehicle reaches the boundary of its capabilities.

Furthermore, more robust technology and unimpeachable image recognition algorithms are required to detect the objects in the surrounding. If the objects are detected learning algorithms can be used to distinguish between which of these objects is an obstacle and which is not. Both human and machine error is possible in this case, therefore more research is needed to put together robust and trustable technology which is capable of warning the human when potential errors are about to happen. The warnings can extend from the driver being distracted to obstacles approaching the vehicle. The car should be able to provide a heads-up warning to the driver at the right times. To address this, in this paper, an embodied agent is used which warns the driver to transfer control. Although it was not tested on the users if the embodied agent can help to successful make a transfer of control, this research could be further extended by conducting a user test on different scenarios when a hand-over or take-over should be initiated.

Some initial ideas for the hand-over part, which could not be further explored due to the restricted amount of time and resources, are shown in figure 7. The concept behind this is to detect a possibly more dangerous state of the driver when driving. The system should be able to detect when a driver is in a tired state or even when he threatens to fall asleep. Suggesting the driver to hand over control since he is incapable of driving responsibly. This is expected to improve the trust relation between the driver and the system considerably. Moreover, assisting the driver when he needs rest would probably lead to a more pleasant experience compared to the warning when distracted. Not only because in the case of distraction a driver might choose deliberately to be distracted and because it might happen more frequently, but especially since the driver would probably will is happy to take some rest. Since the driver is less frustrated by the suggestion it might lead to easier adoption of the system.

For the embodiment of the avatar a more humanlike character was envisioned rather than a static one that only changes color and fills in according to level of certainty. The rationale behind this is that humans are excellent in assessing the current emotional state of a character based on their movement [20]. This also helps in engaging in a more profound social relationship due to more natural reactions. Furthermore, this way it becomes possible to warn the user by the movement and reaction of the avatar rather than presenting him with a more advanced 'certainty meter' [21], as is the case in our final design. To enable a sound and understandable communication of the state of the system it would have been useful to use the 12 principles of animation of Disney [22].

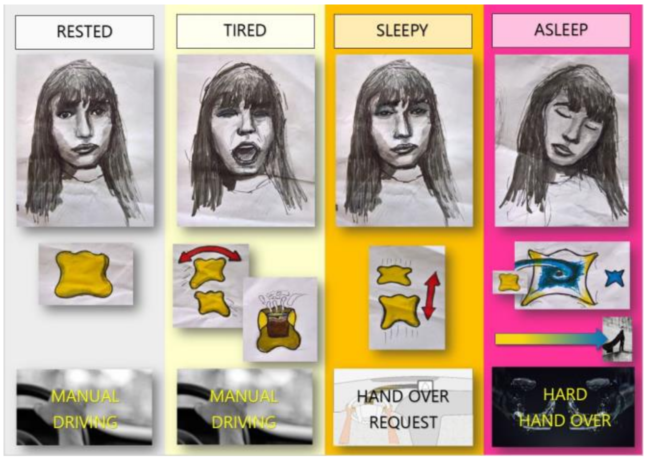


Figure 7: Initial concept for hand-over

To remind the user to be more focussed, the avatar could jump up and down showing how active he is and how much you are not. He could also offer you to guide you to a place to get some coffee, in case you feel like it. If this does not help and the agent arrives at a more alert state it can show the user how excited he is to take over control from the user. It will probably give the user more confidence to give away control to an agent that is jumping up and down to show his eagerness than an agent that warns you to pay attention. In the final stage the 'agent is taking control now' animation could be more noticeable than is the case now. Not only could it turn blue together with the flowing movement of the lights LED strip, the flowing of the 'energy' should continue from the LED strip to the screen into the avatar. Furthermore should transformation of the avatar be more dramatic it should be apparent to the user that it gains the 'superpower' to control the vehicle. Hence the avatar should be more powerful visually and behave less carefree and more formal and precise, being in full control and having the final decision. As stated before, time and resources were the main reasons this vision never fully materialized.

In the second part of this project was also focused on initiating a social interaction between the agent and the users by using a game. Although this idea is quite futuristic, it would help to create a rapport between the passengers in the car and the agent. When the car is in an autonomous mode, passengers can initiate a conversation with the agent by playing games or even asking about the weather or the traffic. The speech recognition module can be further extended to do a lot of other things, where the agent can successfully respond to any queries from the user. Another user test can be conducted to find out if speech based interaction indeed helps to increase trust on the AV, since trusting an AV is the central focus and vision of this project.

6. REFERENCES

[1] Team, T.T.M. Your Autopilot has arrived. 2015 14.10.2015; Available from: http://www.teslamotors.com/blog/yourautopilot-has-arrived.

[2] Wilson, M., 2002. Six views of embodied cognition. *Psychonomic bulletin & review*, *9*(4), pp.625-636.

[3] van der Heiden, Remo, Shamsi T. Iqbal, and Christian P. Janssen. "Priming Drivers before Handover in Semi-Autonomous Cars." *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2017.

[4] Rezvani, Tara, Katherine Driggs-Campbell, Dorsa Sadigh, S. Shankar Sastry, Sanjit A. Seshia, and Ruzena Bajcsy. "Towards trustworthy automation: User interfaces that convey internal and external awareness." In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pp. 682-688. IEEE, 2016.

[5] Donmez, B., Boyle, L.N. and Lee, J.D., 2006. Safety implications of providing real-time feedback to distracted drivers.. *Accident Analysis & Prevention*, *39*, pp.581-590.

[6] Maciej, J. and Vollrath, M., 2009. Comparison of manual vs. speech-based interaction with in-vehicle information systems. *Accident Analysis & Prevention*, *41*(5), pp.924-930.

[7] Parde, N.P., Papakostas, M., Tsiakas, K., Dagioglou, M., Karkaletsis, V. and Nielsen, R.D., 2015, April. I Spy: An Interactive Game-Based Approach to Multimodal Robot Learning. In *AAAI Workshop: Knowledge, Skill, and Behavior Transfer in Autonomous Robots*.

[8] Parde, N.P., Papakostas, M., Tsiakas, K., Dagioglou, M., Karkaletsis, V. and Nielsen, R.D., 2015, April. I Spy: An Interactive Game-Based Approach to Multimodal Robot Learning. In *AAAI Workshop: Knowledge, Skill, and Behavior Transfer in Autonomous Robots.*

[9] Schalkwyk, J., Beeferman, D., Beaufays, F., Byrne, B., Chelba, C., Cohen, M., Kamvar, M. and Strope, B., 2010. Google Search by Voice: A case study. *Advances in Speech Recognition: Mobile Environments, Call Centers and Clinics*, pp.61-90.

[10] Walch, M., Lange, K., et al. (2015). *Autonomous Driving: Investigating the Feasibility of Car-Driver Handover Assistance*, *AutomotiveUI '15* Proceedings of the 7th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, 11-18.

[11] Gold, C., Damböck, D., et al. (2013). *"Take-over!" How long does it take to get the driver back in the loop?*, Proceedings of the Human Factors and Ergonomics Society 57th Annual Meeting, 1938-1942.

[12] Politis, I., Brewster, S., and Pollick, F. (2015). *Language-Based Multimodal Displays for the Handover of Control in Autonomous Cars*, AutomotiveUI '15 Proceedings of the 7th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, 3-10.

[13] R. Parasuraman and C. A. Miller. 2004. Trust and Etiquette in High-criticality Automated Systems. Commun. ACM 47, 4 (April 2004), 51–55.

[14] http://www.telegraph.co.uk/news/2016/07/01/tesla-autopilot-crash-driver-was-watching-harry-potter-movie-wit/ (accessed 06/17)

[15] https://www.tesla.com/presskit/autopilot (accessed 06/17)

[16] Walch. M , Mühl. K, Kraus. J, Stoll.T, Baumann. M and Weber. M , From Car-Driver-Handovers to Cooperative Interfaces: Visions for Driver–Vehicle Interaction in Automated Driving

[17] *V. N. Vapnik, The Nature of Statistical Learning Theory.*

New York: Springer-Verlag, 1995.

[18] Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. The annals of mathematical statistics, 37(6), 1554-1563.

[19] http://psychologia.co/mirroring-body-language/ (retrieved 06/2017)

[20] Hoffman, G., Ju, W. Designing Robots With Movement in Mind , Journal of Human-Robot Interaction., Vol.3, No. 1, 2014

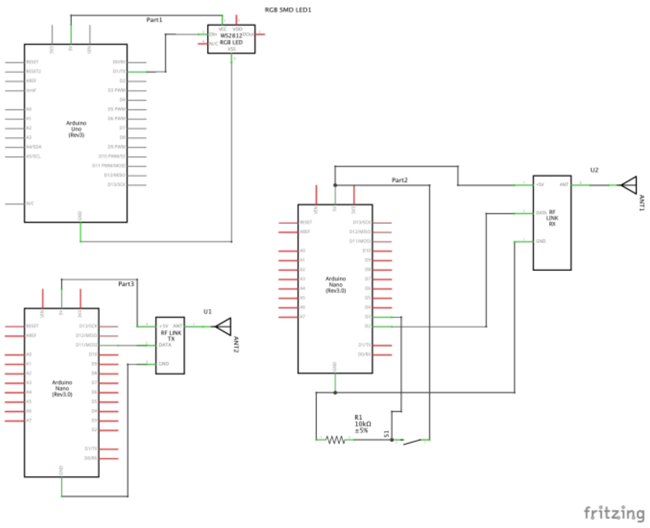
[21] Helldin, T., Falkman G., Riveiro M., Davidsson S., Presenting system uncertainty in automotive UIs for supporting trust calibration in autonomous driving, *Conference: Proc. 5th Int. Conf. on Automotive User Interfaces and Interactive Vehicular Applications (Automotive'UI 13)*, 2013

[22] http://minyos.its.rmit.edu.au/aim/a_notes/anim_principles.html (accessed 05/28)

[23] R.-H. Liang, “howieliang/LibSVM4Processing,” GitHub, 17-May-2017. [Online]. Available: https://github.com/howieliang/LibSVM4Processing/tree/master/e3_Load_data_from_CSV. [Accessed: 10-Jun-2017]

7. Appendix

Appendix A: Schematic of Circuit Design Apparatus.



Appendix B: Code for driver distraction detection and hand-over

Part one : Training

The code for this program was developed in processing.To train the svm model following program was used. This code uses the library LibSVM for processing. The input to the model is a csv file (two.csv) with three columns, two for features and one for class labels. Features are the coordinate values for the eyes As mentioned earlier there are three classes 0,1,2. This code will generate a SVM model that is used in the testing program. Also, part of the implemented code is based on the program provided by Rong-Hao Liang [21].

```
double C = 64;
int d = 2; //feature number

void setup() {
  fullScreen();
  //load CSV file for training and classifying
  trainData = loadCSV("two.csv");
  svmTrained = false;
  firstTrained = false;
}

void draw() {
  background(255);
  if (!svmTrained && firstTrained) {
    //train a linear support vector classifier (SVC)
    trainLinearSVC(d, C);
  }
  //draw the SVM
  if (d == 2) drawSVM();

  if (svmTrained) {
    //form a test data
    double[] testData = {709.9595/800.0,282.0548/800.0};
    print(testData[0],testData[1]);

    int predict = (int) svmPredict(testData);
    println(predict);
    drawPrediction(predict, testData);
  } else {
    drawCursor();
  }
  drawInfo(10, height-124);
}

void drawInfo(int x, int y) {
  String manual = "\n- Press [ENTER] to Train the SVM"+
    "\n- Press N=[0-9] to Train an Linear SV Classifier with
C=2^N"+
    "\n- Press [TAB] to change label color"+
    "\n- Press [/] to clear data"+
    "\n- Press [S] to save model"+
    "\n- Scroll mouse to adjust noise";
  if (firstTrained) {
    trainingInfo = "Linear-Kernel SVM, C = "+ nf ((float)C, 1, 0)
+", In-sample Accuracy = "  + nf ((float)best_accuracy*100, 1, 2)
+ "%"+ manual;
  } else {
    trainingInfo = "Linear-Kernel SVM, C = "+ nf ((float)C, 1, 0) +
manual;
  }
  pushStyle();

  stroke(0);
  noFill();
  rectMode(CENTER);
  rect(width/2, width/2, width-1, width-1);
  fill(255);
  rect(width/2, height-(height-width)/2, width-1, (height-width-1));
  noStroke();
  fill(0);
  textSize(12);
  text(trainingInfo, x, y);
  popStyle();
}


void mouseWheel(MouseEvent event) {
  noise += event.getCount();
  if (noise > width) noise = width;
  if (noise < 1) noise = 1;
}

void mouseDragged() {
  if (mouseX < width && mouseY < height) {
    double px = (double)mouseX/(double)width+ (-
(noise/2)+noise*randomGaussian())/(4*width);
    double py = (double)mouseY/(double)height+ (-
(noise/2)+noise*randomGaussian())/(4*width);
    if (px>=0 && px<=1 && py>=0 && py<=
((double)width/(double)height)) {
      println(px*800,py*800);
      Data newData = new Data(new double[]{px, py, type});
      trainData.add(newData);
      ++tCnt;
    }
  }
}

void keyPressed() {
  if (key == ENTER) {
    if (tCnt>0 || type>0) {
      if (!firstTrained) firstTrained = true;
      resetSVM();
    } else {
      println("Error: No Data");
    }
  }
  if (key >= '0' && key <= '9') {
    C = pow(2, key - '0');
    if (!firstTrained) firstTrained = true;
    resetSVM();
  }
  if (key == TAB) {
    if (tCnt>0) {
      if (type<(colors.length-1))++type;
      tCnt = 0;
    }
  }
  if (key == '/') {
    firstTrained = false;
    resetSVM();
    clearSVM();
  }
  if (key == 'S' || key == 's') {
    if (model!=null) {
      saveSVM_Model(sketchPath()+"/data/test_final.model",
model);
      println("Model Saved");
    }
  }
}
```

Part two : Testing

To test the prototype the following code was implemented. This code accepts user features and the SVM model as input. The user features are detected automatically using kinnect2. The output will be class labels mapped to a specific avatar displayed on screen. Also, when the car requests driver to give control, the led strip blinks and when it is autonomous mode the strip glows continuously. This code was also implemented in processing.

```
import KinectPV2.*;
import processing.serial.*;
import cc.arduino.*;

Arduino arduino;
Serial arduinoPort;
KinectPV2 kinect;

boolean auto = false;
boolean requestAV = true;



int pinFront = 4;
int pinFrontLight = 10;
int pinBack = 5;
int pinBackLight = 11;
int saved;
int tTime=2000;
boolean first_time=true;
boolean light_key=true;
int len=500;
int incr=100;

void setup() {
  fullScreen();
  background(0);
  initialsketch();



  // Load the svm model

model = loadSVM_Model(sketchPath()+"/data/test_final.model");
  svmBuffer = getModelImage(svmBuffer, model, (double)width,
(double)height);
  svmTrained = true;
  firstTrained = false;

  //Initialise the kinnect


   kinect = new KinectPV2(this);
   kinect.enableHDFaceDetection(true);
   kinect.enableColorImg(true); //to draw the color image
   kinect.init();

  //Initialise the arduino


  arduino = new Arduino(this, Arduino.list()[1], 57600);
  String arduinos = Serial.list()[0];
  arduinoPort = new Serial(this, arduinos, 9600);
  for (int i = 0; i <= 13; i++)
    arduino.pinMode(i, Arduino.INPUT);

  // Initiailise timer

  saved=millis();
}

void draw() {
 // this part of code is for buttons and led strip
 if(auto)
 {
    if(requestAV)
    {
      pulse(pinBackLight);
    }
 }
 else
 {
   if(requestAV && (len>=0 && len<300))
   {
     pulse(pinFrontLight);
     delay(1000);
     arduino.analogWrite(pinFrontLight, 0);
     arduinoPort.write('2');

     arduino.analogWrite(pinBackLight, 0);

     arduinoPort.write('1');

     if(len==0)
     {
       print("hello:");
        pulse(pinFrontLight);
     arduino.analogWrite(pinFrontLight, 0);
     arduinoPort.write('2');
     }
   }
}
```

```
  if (arduino.digitalRead(pinBack) == Arduino.HIGH){
    if( auto ){

      auto = false;
       light_key=true;
      arduino.analogWrite(pinBackLight, 0);
      println("User in control");
      arduinoPort.write('1');
    }
  }
  if (arduino.digitalRead(pinFront) == Arduino.HIGH){
    if( !auto ){
      len=0;
      auto = true;
      light_key=false;
      arduino.analogWrite(pinFrontLight, 0);
      println("Autonomous driving");
      arduinoPort.write('2');
      increment();
    }
  }

  int passedTime= millis()-saved;
  float l=0,m=0;
  double[] testData= new double[2];

// code for detecting face
  if(light_key)
  {
ArrayList<HDFaceData>hdFaceData=
kinect.getHDFaceVertex();
   for (int j = 0; j < hdFaceData.size(); j++)
   {
      //obtain a the HDFace object with all the vertex data
      HDFaceData HDfaceData = (HDFaceData)hdFaceData.get(j);
      if (HDfaceData.isTracked())
      {

        for (int i = 0; i < KinectPV2.HDFaceVertexCount; i++)
        {
          if((i>100 && i<120) || (i>1090 && i<1093)  )
          {
            float x = HDfaceData.getX(i);
            float y = HDfaceData.getY(i);

            if(i==101)
            {
              l=x;
              m=y;
            }

          }
        }
        }
    }

     //form a test data
    if(passedTime >tTime)
    {
      testData[0]=(double)l/800.0;
      testData[1]=(double)m/800.0;
      saved=millis();


      if (l>0)
     {
        // predicting the result
        int predict = (int) svmPredict(testData);
        println(testData[0],testData[1],predict);
        if(predict==0)
        {
          decrement();
        }
        else
        {
          increment();
        }
      }
      }

   }

}

// this part makes the led turn on/off
int bright = 0;
int step = 15;
int dir = 1;
void pulse(int pin) {

  arduino.analogWrite(pin, bright);
  bright = bright + step * dir;
  if (bright >= 255 || bright <= 000){
      dir = dir *-1;
  }
}

//Functions for drawing and manipulting the agent
// increment has the code for when driver is not focused on driving
void increment()
{
  if((len-incr)<=0)
    {
      len=0;
      stroke(187,10,30);
      fill(#64B5F6);
      star(3, displayWidth/2, displayHeight/2, 500, 500, -PI / 2.0,
0.35);
      //handover
      auto = true;
      light_key=false;
      arduino.analogWrite(pinFrontLight, 0);
      println("Autonomous driving");
      arduinoPort.write('2');
      len=500;
    }
    else
    {
      stroke(187,10,30);
      fill(234,26,62);
      star(3, displayWidth/2, displayHeight/2, 500, 500, -PI / 2.0,
0.35);

      len=len-incr;
      stroke(187,10,30);
      fill(0);
      star(3, displayWidth/2, displayHeight/2, len, len, -PI / 2.0,
0.35);
    }
}
```

```
void decrement(){ // code for when driver is focused on driving

  if ((len+incr)>=500)
    {

      len=500;
      stroke(187,10,30);
      fill(0);
      star(3, displayWidth/2, displayHeight/2, 500, 500, -PI / 2.0,
0.35);
    }
    else
    {
      stroke(187,10,30);
      fill(234,26,62);
      star(3, displayWidth/2, displayHeight/2, 500, 500, -PI / 2.0,
0.35);
      len=len+incr;
      stroke(187,10,30);
      fill(0);
      star(3, displayWidth/2, displayHeight/2, len, len, -PI / 2.0,
0.35);

    }
}

void initialsketch(){
  stroke(187,10,30);
  fill(0);
  star(3, displayWidth/2, displayHeight/2, 500, 500, -PI / 2.0,
0.35);

}

void star(int n, float cx, float cy, float r, float prop) {
  star(n, cx, cy, 2.0 * r, 2.0 * r, 0.0, prop);
}

void star(int n, float cx, float cy, float w, float h,
  float startAngle, float proportion) {

  if (n > 2) {
    float angle = TWO_PI/ (2 *n);  // twice as many sides
    float dw; // draw width
    float dh; // draw height
    w = w / 2.0;
    h = h / 2.0;

    beginShape();
    for (int i = 0; i < 2 * n; i++) {
      dw = w;
      dh = h;
      if (i % 2 == 1) {
        dw = w * proportion;
        dh = h * proportion;
      }
      vertex(cx + dw * cos(startAngle + angle * i),
        cy + dh * sin(startAngle + angle * i));
    }
    endShape(CLOSE);
  }
}
```

Appendix C. Code used to create the prototype I-spy game.

```
import speech_recognition
from gtts import gTTS
import os
import sys
import time
import random
import pyglet

recognizer = speech_recognition.Recognizer()

def sleep():
    time.sleep(3)

def runalltime():
    while True:
        pickgame()


def pickgame():
    say("What would you like to do today?")
    time.sleep(2)
    reply=listen()
    reply=reply.lower()
    reply="I would like to play a game" #add listen here
    if "game" in reply:
            data3= "That is great"
            data4= "In my list I have I spy"
            data5= "Would you like to play?"
            say(data3)
            time.sleep(2)
            say(data4)
            time.sleep(2)
            say(data5)
            time.sleep(2)
            say("Awesome")
            time.sleep(2)
            say("Let us play I spy")
            time.sleep(2)
            reply= "yes"
            reply=listen()
            reply=reply.lower()
            print (reply)
            if "yes" in reply:
                game()

            else:
                game()

def game():
    spy_list = ['car','tree','sky','bird', 'human']

    say("Do you know the rules of the game?")
    time.sleep(2)
    reply=listen()
    reply=reply.lower()
    if "no" in reply:
            data3= "There is something on my mind"
            data4= "With my hints you guess what it is"
            say(data3)
            say(data4)
            time.sleep(2)

    say("Are you ready to play?")
    sleep()
    reply="yes"
    reply=listen()
    reply=reply.lower()
    print(reply)
    while "yes" in reply:
        say("I spy with my little eye something that starts with an S
        and is blue")
        #sleep()
        reply="Sky"
        reply=listen()
        reply=reply.lower()
        print(reply)
        if sky in reply:#if spy_list[random_value] in reply:
            say("Good Job, You win")
            sleep()
            break
        else:
             say("Wrong Guess, Do you wanna guess again")
             sleep()
             reply="yes"
             reply=listen()
             reply=reply.lower()
             if "yes" in reply:
                 say(" the second clue is, Look up and you see what I
        see")
                 reply=listen()
                 reply=reply.lower()
                  print(reply)
                 if "sky" in reply:
                     say("Good job you win")
                     sleep()
                     say("do you want to continue?")
                     reply=listen()
                     reply=reply.lower()
                     reply="no"
                     if no in reply:
                         say("Okay!")



             break


def listen():
    with speech_recognition.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)
    try:
        #return recognizer.recognize_sphinx(audio)
        return recognizer.recognize_google(audio)

    except speech_recognition.UnknownValueError:
        return "Could not understand audio"
    except speech_recognition.RequestError as e:
        return "Recognition Error"
    return ""
def say(data):
    tts = gTTS(text=data, lang='en')
    tts.save("out.mp3")
##    os.system("mpg321 out.mp3 -quiet")
    os.system("start out.mp3")
    sleep()
    os.system("TASKKILL out.mp3")
    os.remove("out.mp3")
##    tts = gTTS(text='Hello World', lang='en')
##    tts.save("out.mp3")
##
##    music = pyglet.media.load("out.mp3", streaming=False)
##    music.play()
##
##    sleep(music.duration) #prevent from killing
##    os.remove("out.mp3") #remove temperory file
##



def main():
    try:
       while True:
           reply="Lisa"
           reply=listen()
           reply=reply.lower()
           print (reply)
           input_user="Lisa"
           input_user=listen()
           input_user=input_user.lower()
           if "a" in input_user:

               #invoke function called options

               data1="Hi there, I am Lisa"
               sleep()
               say(data1)
               sleep()
               runalltime()
               time.sleep(2)
                 say("Bye")
               break
           else:
               print (input_user)
               data="Sorry, did not get you"
               say(data)
               say(input_user)
               break
               #print(input_user)
       #runalltime()
    except KeyboardInterrupt :
        sys.exit()

if __name__ == "__main__":
    main()
```