Applying Hybrid Dynamic Logic for use in Game Development

15-424 Final Project
By Woody McCoy

Abstract

Hybrid dynamic logic and hybrid systems are a relatively new field that is focused on modeling continuous, dynamic systems and proving their safety. While these systems have found a wide variety of use already in fields like robotics and autonomous vehicles, their use cases extend far beyond what has already been explored.

This project investigates the use of hybrid systems and hybrid dynamic logic for use in game development. The reasons for this are twofold. Firstly, many portions of game development are extremely analogous to the types of systems that hybrid dynamic logic normally models. Many games feature autonomous agents that must traverse a complicated world intelligently, and must do so while fitting the constraints of only being able to interact with the world once per frame. This common use case corresponds exactly to a standard time-based controller in hybrid dynamic logic, and shows that a clear parallel exists. While this is one simple example, many other similar cases exist that could also be explored through hybrid systems.

Another feature of hybrid dynamic logic that could be applied to games is the ability to prove the safety of a system. Many games struggle to prove the safety of their content, with safety being that a system is fair and fun to engage with. This is especially true with games that use randomization to generate content, and must attempt to balance their content on-the-fly. Hybrid systems provide a way to investigate all possible versions of a system, and prove the safety of those versions all at once. This kind of provable safety is incredibly valuable, and could solve longstanding problems in game design.

Hybrid systems are complicated to design and prove, though, which conflicts with the normal flow of a game production pipeline. This project investigates their use by modeling, proving, and implementing two connected systems: an autonomous agent that must survive in a randomized environment, and a dynamic balancing system that attempts to make the game fair for that agent. In modelling these systems, this paper

will investigate the way that hybrid dynamic logic could fit into a production pipeline, and will attempt to draw fair conclusions about its use cases in a real development cycle.

Introduction

Hybrid dynamic logic, and hybrid systems, are an emerging concept in the world of computer science, robotics, and logic that focus on modeling and proving the safety of complex systems and controllers. Hybrid dynamic logic focuses especially on modeling dynamic, evolving systems, which can react nondeterministically to given inputs. This makes it especially powerful, as it is capable of modelling systems in a way that corresponds to how real systems play out. It also has the benefit of proving all possible versions of the system simultaneously, which allows for simple safety proofs to extend to the entire space of the system. While hybrid dynamic logic is already seeing use in fields such as robotics, autonomous vehicles, and computer science, it stands to reason that its use cases could extend far beyond these.

Video games might seem like an initially odd choice for hybrid systems, but they are a perfectly matched use case. The reason for this is a modeling paradigm that has become common in modern games - the use of engine physics. In general, a vast majority of games use a fixed timestep for their physics, which operates independently of the normal control loop. There are a variety of good reasons to do this, the most prominent being that fixed timesteps make physics simulations more accurate, and resistant to frame rate variation. The downside of this, though, is that physics-based entities in the game can't control themselves in between frames.

As games have expanded in scope, they have also moved towards a concept of full interactability - where everything in the game can be touched, moved, thrown, etc. While this greatly enhances immersion, it also requires that nearly every object in the game be physically simulated. From a design standpoint, the simplest way to achieve this is to make all forms of movement, animation, and control physics-based instead of manually controlled. This brings us to a very modern paradigm of game development, wherein all objects use their update sequence every frame to make decisions and set their physics, and then let the game engine physics take over for the fixed updates in between each frame. This is an ideal use case for hybrid systems because it is a one-to-one representation of how hybrid systems model situations - an instantaneous decision making step, followed by a continuous physics step that runs for a nondeterministic amount of time.

Aside from modeling and development, hybrid dynamic logic also stands to solve problems on the game design side as well. Difficulty balancing is a notoriously difficult and fairly unsolved problem in the field, the general answer to which is trial and error. This is due to the large amount of content that is found in games, as well as the

complexity of the interactions that can exist. Hybrid dynamic logic could be used to tackle both of these problems simultaneously, as it can model all possible values at once, as well as model complex interactions. This modelling is especially relevant in the area of dynamic difficulty balancing, wherein an autonomous system in the game attempts to keep the game fair for the player. Modelling this sort of system as a hybrid system seems like a perfect pairing, and brings the full power of hybrid dynamic logic into a previously unsolved problem.

Hybrid dynamic logic may not be perfectly applicable to game development, though. Hybrid systems are very complex, and require a lot of work to accurately model and prove. This conflicts with the general speed at which a game development pipeline wants to be producing content, and so there is a fundamental question of whether the benefits provided by hybrid systems are worth the price of their development.

This project will investigate this problem by creating a small game environment based off of two hybrid systems. The first is an autonomous player that attempts to navigate through a world with randomized enemies and objectives. The second is a dynamic difficulty balancing system, which attempts to use rough heuristics to always maintain a fair game environment. In modelling, proving, and then implementing these systems, the project will explore the actual feasibility of using hybrid systems in games, as well as the usefulness of the actual application of hybrid systems.

Related Work

While there is not a vast amount of overlap between hybrid systems and game design, there is ongoing research that covers some of the niches between two. Firstly, human computer interaction is an ongoing topic of discussion with respect to hybrid systems (Sadigh), and has a vast significance in the way that the robots and programs of the future will interact with their users (Sadigh, Lore et. al). Another overlap is in the study of artificial intelligence, with hybrid systems providing safety to systems that are otherwise very challenging to prove (Choradacho and Lees). Lastly, there is also ongoing research into using hybrid systems to verify decision trees (Ashok et. al). This is a commonly used pattern for both robotic controllers, as well as game AI. Hybrid systems have also made use of games, with simulated environments being used to show the efficacy of controllers for systems such as autonomous vehicles (Lin et. al). This project does not further pursue the value of games to hybrid systems, but rather investigates the reverse, which is a perspective that has not been explored yet.

Model 1 - AI self-preservation

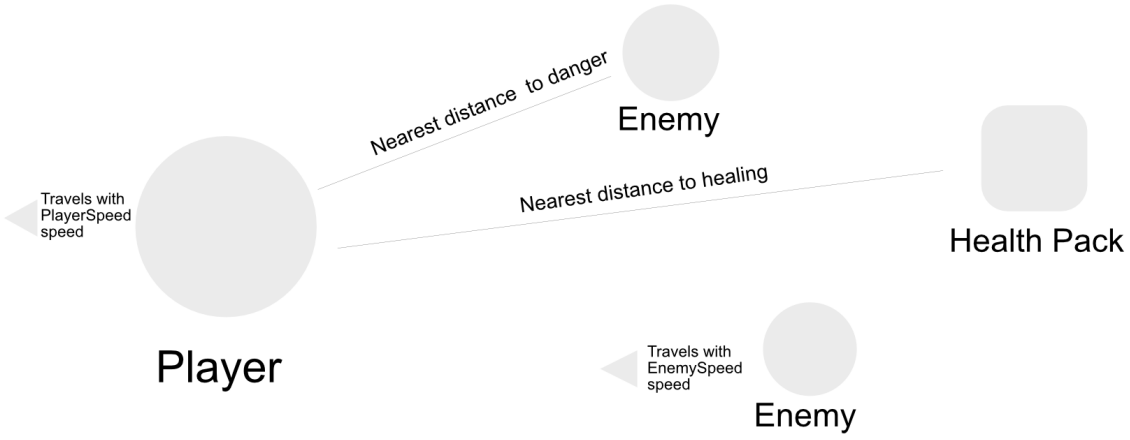

Figure 1: A diagram of the modeled scenario

The first model is a hybrid system that attempts to capture the decision-making and behaviour of an autonomous player in a randomized environment. This kind of artificial intelligence system is fairly common in games - it is used both to make intelligent entities in the world, like enemies and other non-player characters, as well as to perform automated testing of the game environment and mechanics. Since this sort of decision making and movement is very analogous to existing hybrid controllers, it serves as a good starting point for investigation.

The world of the model is as follows: there is a single player entity, which exists somewhere in the space. The goal of the player entity is to ensure that it does not die, which it does by maintaining its health above some threshold. Along with the player entity, there is an opposing enemy entity, whose goal is to get close to the player and deal damage to them. There is also a health pack entity, which cannot move, and returns some amount of health to the player when they reach it. Attacks in the game require the attacker to sit still, and maintain a short distance between their target and themselves in order to complete the attack. This is a simplification of standard attack systems, but does accurately match the normal behavior of most attack systems, wherein an enemy must stop to attack, giving the player time to dodge.

This model features a number of other simplifications which are designed to make it easier to prove the accuracy of the model, without reducing the effectiveness of the model itself. The first simplification is that all entities are considered in distance only, rather than in 2D coordinates. The reasoning for this is that the 2D coordinates are significantly more complicated to prove, but don’t add much more nuance than 1D

distance does. The 2D cases can be safely composed by only considering the 1D version of the worst case, where the player must move through the enemy and suffer damage in order to reach the healing. By enforcing a simplification where the player can only take this move in order to reach the healing, we can know that our AI will work correctly in the worst case, and still have very acceptable behaviour in the best case. This is because it only takes a given move when it would have taken the move regardless of a possible penalty. This simplification makes our model slightly less efficient, as the AI will sometimes forgo clearly safe choices, but at the benefit of guaranteeing its safety.

Another simplification is the existence of only 1 enemy, and 1 health pack. The reasoning for this is that it firstly simplifies the core logic of the AI into 3 choices - to run away, attempt to fight, or attempt to get more health. In all 3 of these cases, it is generally better to prioritize a closer objective from a farther one, as that increases the uncertainty with which the situation might have evolved by the time the player reaches their objective. In the case of the health packs, this simplification is very easy to justify - they are not moving, and so only considering the most relevant entity is a fair simplification. For enemies, this simplification is less easy to justify, and is somewhere the model could be improved. With that being said, considering only the nearest enemy does give good behaviour in most cases, especially as enemies will tend to clump together as the player moves around. The nearest enemy also tends to be the largest threat, as it will be the first entity that is able to damage the player.

For the proof of this system, we model the system with two parts - a control phase followed by a continuous evolution phase. In the control phase, the player picks between the 3 different options available to it. It can choose to run from the enemy, attack the enemy (if the enemy is in range), or move towards the healing. In all cases, there are safety and efficiency checks to make sure that the moves are valid. For fighting, the player can only choose to fight when they have enough health to stay above their red line over the duration of the fight. Otherwise, they are forced to run away. For healing, the player ensures that they could take damage from enemies the entire way to picking up the health pack. In this way, they cover for other enemies being in the scene that are not modeled, and having those enemies attack the player as they move. It’s important to note that a running away option is always available to the player. That way, the player always has at least one valid move to make, and they can filibuster the game if necessary. While this trivializes the safety check for fleeing, it matches the real behavior of the game, where a player can continuously run from an enemy to avoid their attacks.

For continuous evolution, there are 3 different ODE’s that can be run. The first is for when the player is not in combat, and it models the fleeing behavior. It also catches the filibuster case, where the enemy cannot deal damage. It is safe to model the filibuster here, because the fighting ODE would short-circuit immediately when given the

same parameters. The second and third ODE's are tied together, and form an event-triggered system. These are the fighting ODE's, and they model the combat between the player and the enemy, and stop evolving when the player's health reaches the danger threshold. Because of this event trigger, the third ODE is never actually run, but is left in to ensure that the modelling is valid.

After running the control sequence and ODE's, we can prove that the model always maintains its health above the danger threshold, and therefore stays alive. This model could certainly be improved by adding logic for multiple enemies and health packs, but it does do sufficiently well with the simplified case, as will be discussed later in the implementation.

Model 2 - Dynamic Difficulty Balancing

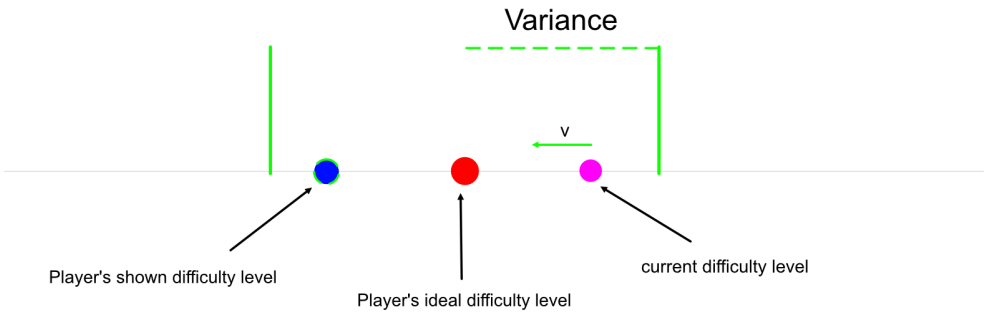

Figure 2: a one-dimensional view of difficulty scaling

The second model is a hybrid system that performs dynamic difficulty balancing. The system is structured around trying to match an ideal difficulty value of a player, but working around a constraint that the actual value cannot be known. Instead, a heuristic value is used that is guaranteed to be within some variance of the ideal value. The system uses this heuristic to determine how to update the current difficulty, and attempts to ensure that the current difficulty is also always within the variance value of the ideal difficulty. In this way, the system does the best that it can with imperfect information. Additionally, the variance and ideal value are both allowed to be in motion, with the variance tightening (in the event that the heuristic could improve with more information over time) and the ideal value moving in any direction.

The hybrid system makes a few simplifications, mostly with the value generated by the heuristic. The new heuristic value is guaranteed to be within the range of the new bounds and the old bounds, which is theoretically not true for a rapidly changing player and environment. However, in general, players tend to acquire skill very slowly, and so there should generally be an overwhelming amount of overlap between the old valid states and new valid states. Furthermore, having this simplification simplifies the logic of the proof into mostly linear systems, which makes proving the system much easier.

The actual proof of the system relies on a control phase, followed by a dynamics phase. In the control phase, a new heuristic value is generated, and the system sets a velocity for the current difficulty to match that heuristic. Once this is done, the system switches to the continuous dynamics phase. In this phase, the current value, ideal value, and variance value are all allowed to follow their respective velocities. The dynamics are split into two cases, in order to capture the event where the current value becomes equal to the shown heuristic value. This split is important, in order to ensure that the system does not overshoot the correct value and exit the safe area.

For the actual proof, we rely on solutions to the ODE's. The solutions for these ODE's are linear, and by enforcing that the velocity of the changing difficulty is higher than the velocity of the ideal difficulty and variance we can ensure that the current difficulty is always changing faster than the borders of the area that it must stay within. This assumption is a simplification, but like before, it is fair because the real systems tend to evolve very slowly, while the speed of the difficulty change can be set arbitrarily high.

The invariants of this system are that the shown heuristic value remains within variance, as well as the current difficulty value. The heuristic should remain with variance by definition of what the heuristic should represent, so this invariant ensures that. By enforcing that the current difficulty also remains within the variance, we can ensure that the end result of being within variance is also true.

Implementation

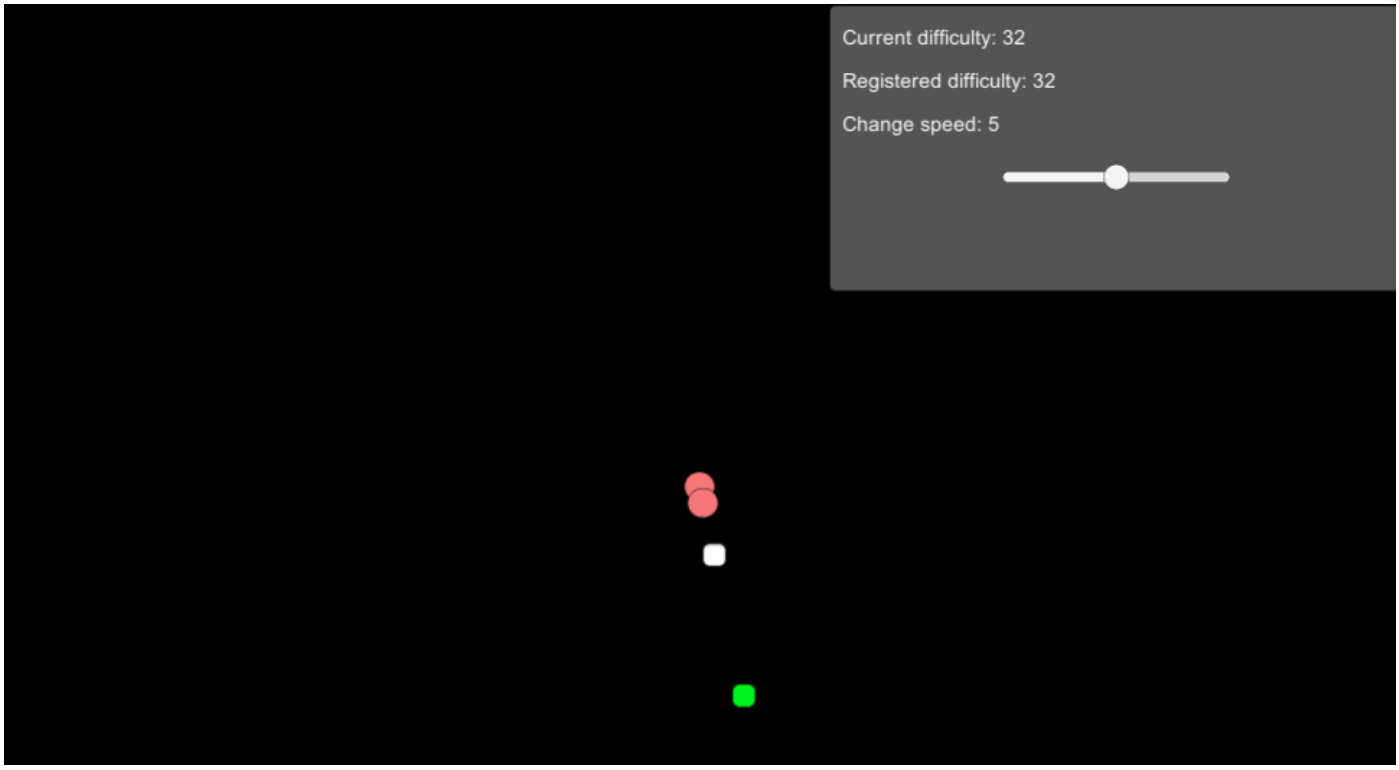

Figure 3: the implementation in action, with the player shown in white, enemies in red, and health packs in green.

Once the models were finished and proven, I implemented the systems they described in Unity. For the most part, the implementation version of the systems was one-to-one with the control scheme dictated by the models. While it would have been possible to abstract the control scheme of the model more into an analogous control scheme, I wanted to really test the accuracy of the exact model in a real game environment.

The implementation consists of a few key files, with some helper files that enable the environment to run. The largest and most complicated file is ‘AI.cs’. This file describes the behaviour of the player character, and attempts to capture the exact behaviour of the autonomous player model (model 1). The AI mimics the model behavior by having 4 predefined behaviors, which correspond to the 4 ways that the model can choose an input and ODE. These behaviors are chosen nondeterministically, and gated behind checks that correspond to the same checks in the model. In this way, the AI approximates the nondeterminism of the hybrid system when choosing a course of action.

The first behavior is a fleeing/approaching behavior, which exactly copies the first case of model 1, where the player chooses to run towards or away from the enemy. The second behavior is a fighting behavior, where the AI attacks an enemy that is within

range. The third case is a fleeing-only behavior, which matches the combat-fleeing behavior in the model. The last case searches for nearby health packs and attempts to pick them up, following the same heuristics as the model.

These behaviors match the behaviors in the model, but have a few key differences. Firstly, they perform heuristics in 1D space but move and perform other operations in 2D. They also do not rely on set values, like the model. Instead, they use the values set by the difficulty balancing system to determine what the correct course of action is. These values are constantly changing as the AI performs better or worse, as will be discussed below.

Alongside the AI, there is also an enemy behavior in “Enemy.cs”. The enemy is a much simplified version of the player AI, and has only a single behavior. Enemies choose to run towards the player at their maximum speed, and attempt to attack the player when the player is in range. If the player escapes their range before the attack is completed, it exits early, and the AI continues to chase the player. The damage of the enemy attacks, as well as the number of enemies that are allowed to exist in the game are both determined by the difficulty balancing system.

The last main entity from the model is the health pack. These are very simple, and exist to do a collision check with the player. When the player AI collides with a health pack, the health pack is deleted and some health is given to the player. The number of health packs and their healing value are also determined by the difficulty balancing system, but there is guaranteed to always be at least one health pack on the field.

Aside from these main behaviors, there are a number of helper classes in the game, which mostly serve to keep the simulation running smoothly and improve the user experience. These systems perform tasks like randomly spawning enemies and health packs, tracking the player position with the camera, and creating the UI windows that allow the player to interact with the simulation variables. None of these classes derive from anything in the models, but they are fairly adjacent to the modeled behavior and should not disrupt the accuracy of the simulation.

An implementation of the difficulty balancing model is also in the game. This model relies on the hybrid system proof that following a simple heuristic will give generally good results. The heuristic for difficulty used by the model is the player’s health divided in half, which was chosen arbitrarily. Since the player’s health is the main goal of the AI, as well as that of the enemies, it serves as a good measurement for the success of the model. Furthermore, the accuracy of the heuristic does not need to be great to achieve good results, as shown by the model. Having a heuristic that is within a reasonable range of the right answer should give reasonable results.

In order to test the effectiveness of the difficulty balancing, the speed at which the current difficulty moves to match the heuristic is exposed to the UI. This way, the viewer of the model can edit this value in real time, and see how the simulation performs

differently. According to the model, the speed at which the difficulty updates must be faster than the speed at which ideal value and heuristic are changing. Since it is possible for the heuristic to rapidly change as the player takes damage, I found that a value of around 5 units per second was ideal for the difficulty change speed.

Overall, the simulation works fairly well, and does show an effective use of the hybrid systems in a game environment. Due to randomness the system does sometimes in unpredictable ways, and might find itself in a configuration that draws the player away from the main game area. In general, though, the player effectively fights enemies and maintains a high level of health. The difficulty balancing system tends to stabilize in the 30-40 range as well, which is indicative that a correct level of difficulty has been reached.

To interact with the game system, you'll need to launch the attached executable, which will start the simulation. Once the game is running, you can use the provided window to view the game environment, and manipulate the variables of the environment using the UI. If you would like to see more into the state of the game, you can also launch the included project in Unity 2020.3.17f1, which will provide an expanded view of the game, as well as access to all variables that control the game state.

Conclusion

Overall, the project was successful in completing its modeling and implementation goals. A full model for AI self-preservation and a model for dynamic difficulty balancing were both made, proved, and then translated into an actual implementation. The implementation runs an accurate simulation of the models, and shows off the way that the dynamics work in practice rather than in theory.

There is still lots of room for improvement on the project, especially in terms of increasing the scope and modeling power of the hybrid systems. One of the original goals of the project was to use more than one enemy in the first model, in order to capture the complicated behaviors that can come up when the AI is surrounded by enemies on all sides. Unfortunately, this goal ended up being out of the scope of the project, and was not completed in time. While the existing behavior performs well enough for the implementation, improved behavior on both the modelling and implementation side would have taken the project further.

The implementation was quicker to implement than expected, which is a success of the practice of working off of a hybrid system. Both models translated incredibly well into code, which made it very easy to get an initial version of the implementation running. Additionally, the simplified nature of the model's safety checks made it easy to implement some of the more complex behaviors found in the implementation. For instance, rather than performing some sort of weighted check against all the enemies in the scene, the list of enemies could be sorted instead and simple checks that the model

has can be used to determine the correct behavior. Having this simplicity yet knowing that the safety from the model is preserved was a good paradigm to work in, and could certainly be extended to other games.

Similarly, the automatic difficulty balancing was quick to implement thanks to the existing model. The implementation does appear to work well, which offers credence to the idea that this type of model and translation could be used effectively in other games. The general applicability of the model means that it could be directly integrated into many types of games, or be easily modified to account for more specific factors in a given game's balance. Especially in terms of design, the safety of this kind of paradigm seems to be well worth the modeling and proving effort.

The most notable downside to using hybrid systems in a game development pipeline is the speed of developing and proving the models. While the hybrid systems markedly improved the speed of development on the game itself, they required a significant amount of time to design and prove. In a real development environment and with the current tooling, it would most likely be impractical to model and prove safety for all elements of a game. Specifically proving the safety of key designs, though, still has a lot of potential and was well worth the cost in this project. There exists a need that hybrid systems could fill, and with more specific and dedicated tooling they could find industry use.

Overall, this project finds that hybrid systems are very applicable to game development. They can accurately model and prove many key systems for games, and translate well into code. The speed of development is a major blocking point, and is currently too slow to see use in a real development environment. Regardless, the benefits to safety in design are incredibly good, and do warrant the time for a careful designer. With more investment into dedicated solutions and tooling for these kinds of problems, hybrid dynamic logic could provide answers to previously unsolved problems in game design, and could likely find widespread adoption. Hopefully, this concept will be explored more by other investigations and applications in the future.

Sources

1. Rodríguez, Juan & Lees, B. (1998). “ProBIS: Modelling intelligence with hybrid systems.”
2. Dorsa Sadigh, “Human-CPS Through the Lens of Learning and Control.” International Conference on Hybrid Systems: Computation and Control April 21-24, 2020, Sydney, Australia. Keynote Address.
3. K. G. Lore, N. Sweet, K. Kumar, N. Ahmed and S. Sarkar, "Deep Value of Information Estimators for Collaborative Human-Machine Information Gathering," *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, 2016, pp. 1-10, doi: 10.1109/ICCPS.2016.7479095.
4. Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Křetínský, Maximilian Weininger, and Majid Zamani. 2020. DtControl: decision tree learning algorithms for controller representation. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control* (*HSCC '20*). Association for Computing Machinery, New York, NY, USA, Article 17, 1–7. DOI:https://doi.org/10.1145/3365365.3382220
5. Q. Lin, S. Mitsch, A. Platzer and J. M. Dolan, "Safe and Resilient Practical Waypoint-Following for Autonomous Vehicles," in *IEEE Control Systems Letters*, vol. 6, pp. 1574-1579, 2022, doi: 10.1109/LCSYS.2021.3125717.