Design specification

December 2021

Version 1.0

Status

Reviewed	Björk, Rasmus	2021-10-15
Approved		

Toyota Material Handling
carhe007@student.liu.se

Project Identity

Group E-mail:	carhe007@student.liu.se
Homepage:	https://tsrt10.gitlab-pages.liu.se/2021/toyota/
Orderer:	Anton Kullberg Phone: Not applicable E-mail: anton.kullberg@liu.se
Customer:	Oskar Bergkvist (Toyota Material Handling) Phone: Not applicable E-mail: oskar.bergkvist@toyota-industries.eu
Supervisor:	Hamed Haghshenas Phone: Not applicable E-mail: hamed.haghshenas@liu.se
Course Responsible:	Daniel Axehill Phone: Not applicable E-mail: daniel.axehill@liu.se

Project members

Carl-Hampus Hedén	Project manager	carhe007@student.liu.se
Mahdi Najafi	-	mahna987@student.liu.se
Alfed Boman	Head of design	alfbo741@student.liu.se
Adam Kagebeck	-	adaka206@student.liu.se
Kalle Blomkvist	Head of software	karla625@student.liu.se
Rasmus Björk	Head of documentation	rasbj268@student.liu.se
Viktor Ekström	Head of testing	vikek514@student.liu.se

CONTENTS

Toyota Material Handling
carhe007@student.liu.se

DOCUMENT HISTORY

Version	Date	Changes	Made By	Reviewed
0.1	2021-10-04	Draft of the design specification	All	Rasmus Björk
0.2	2021-10-11	Changes according to feedback from supervisor	All	Rasmus Björk
0.3	2021-10-15	Changes according to feedback from orderer	All	Rasmus Björk
1.0	2021-12-14	Final version	All	Mahdi Najafi

1 INTRODUCTION

Toyota M.H. have recently launched their fully autonomous vehicles after having previously used autonomous systems on the existing trucks. The control system for the new AGV is developed by an external company but Toyota M.H. have the ambition to, in the future, develop the controls for the vehicle in house. Based on a Master's thesis work [1] it has become apparent that tweaking the control system for different environments is excessively time consuming. As the simulated working environment is sub-optimal to the actual working environment, the final tuning will be finished on a customer to customer basis. Previously, the control system from the simulation has been tuned to an estimated 90% of the desired controller. The goal of this project is to increase the controller's performance to 95%. Therefore, Toyota M.H. in cooperation with Linköping University created this project to explore the possibilities of automating the process by using machine learning to tune the controllers.

This document aims to describe the design of a system that can be used to evaluate if two different reinforcement learning methods, Proximal Policy Optimisation (PPO) [2] and Deep Deterministic Policy Gradient (DDPG) [3], can be used to automate the control tuning process. A general description of the system is given in Section 2, showing a schematic view of the complete system together with a brief description of its subsystems. These subsystems are then described further in the remaining sections starting from Section 3.

2 GENERAL DESCRIPTION OF THE SYSTEM

The complete system consist of the following subsystems:

- A *controller* that is responsible to steer the AGV using a reference path and measurements of the AGV's physical states.
- A *scenario generator* that generates reference paths as well as parameters for various disturbances that will be modeled in the controller- and AGV model component.
- An *AGV model* that model the physical behaviour of the AGV as closely as possible and output the physical states and measurements of those states.
- An *auto tuner* that take observations and evaluations of the AGVs behaviour as input and outputs control parameters.
- An *observation generator* that generate these observations from the reference path, control signals and measurements of the physical states
- An *evaluator* that generate evaluations of the AGVs behavior from the reference path and measurements of the AGVs physical states.

Figure 1 shows a schematic of the complete system with their respective inputs and outputs.

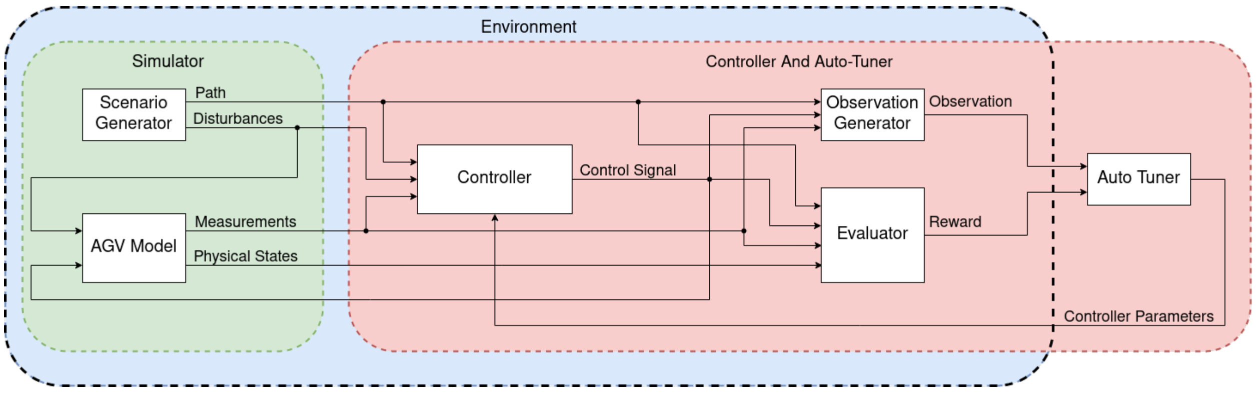

Figure 1: A schematic view of the complete systems components together with their respective inputs and outputs. The red area mark the area that would be implemented in the physical AGV. In that case the green area would be exchanged with the physical AGV. The blue area marks the environment with which the auto-tuner interacts.

3 CONTROLLER

A large group of controllers is compatible with the auto-tuners evaluated in this project. The first controller that will be evaluated in this project is a PID controller and if time allows an LQ controller will also be evaluated.

3.1 Interface

Table 1 show the output of the controller component and Table 2 show the input to the controller component.

Table 1: The output signals of the controller

Data Bus Name	Signal Name	Data Type	Unit
Control Signals	Left Motor Reference Velocity	Float	m/s
	Right Motor Reference Velocity	Float	m/s

Table 2: The input signals of the controller

Data Bus Name	Signal Name	Data Type	Unit
Control Signals	Heading velocity	Float	m/s
	Heading angle	Float	rad (counter clockwise positive)

3.2 PID

In this section the PID controller is described. The input- and output signals to the PID controller are presented in Table 1 and 2. To determine which input signal that will control which output signal, an RGA-analysis [4] will be performed. Since the controller will have two input signals and two output signals, decoupled control will be used [4]. With the decoupled control, a change of variable will be performed where $\tilde{y} = W_2 y$ and $\tilde{u} = W_1^1 u$. This results in a transfer function

$$\tilde{G} = W_2(s)G(s)W_1(s) \tag{1}$$

that is as near a diagonal matrix as possible. Here, G is the original transfer function of the system. This will result in a regulator described below.

$$u = -W_1 F_y^{diag} W_2 y \tag{2}$$

where F_y^{diag} is the PID controller where each input will control each output separately. The matrices W_1 and W_2 will be designed as:

$$W_1 = G^{-1}(0)$$

$$W_2 = I$$

where I is the unit matrix and G^{-1} is the inverse of the kinematic model/system.

4 AUTO TUNER

The Auto Tuner's main task is to provide the control system with controller parameters using Machine Learning (ML) algorithms and will consist of two major building blocks. The first block is a Reinforcement Learning (RL) algorithm that uses rewards and observations to update the policy, and the other one is a policy function that calculates the AGV's controller gains. Figure 2 shows a schematic sketch of Auto Tuner's structure.
In this project, two different RL algorithms will be used to construct the Auto Tuner. The first method is called Proximal Policy Optimization (PPO) and the second method is Deep Deterministic Policy Gradient (DDPG). A detailed description of the two algorithms is given in Section 4.3 and 4.4.

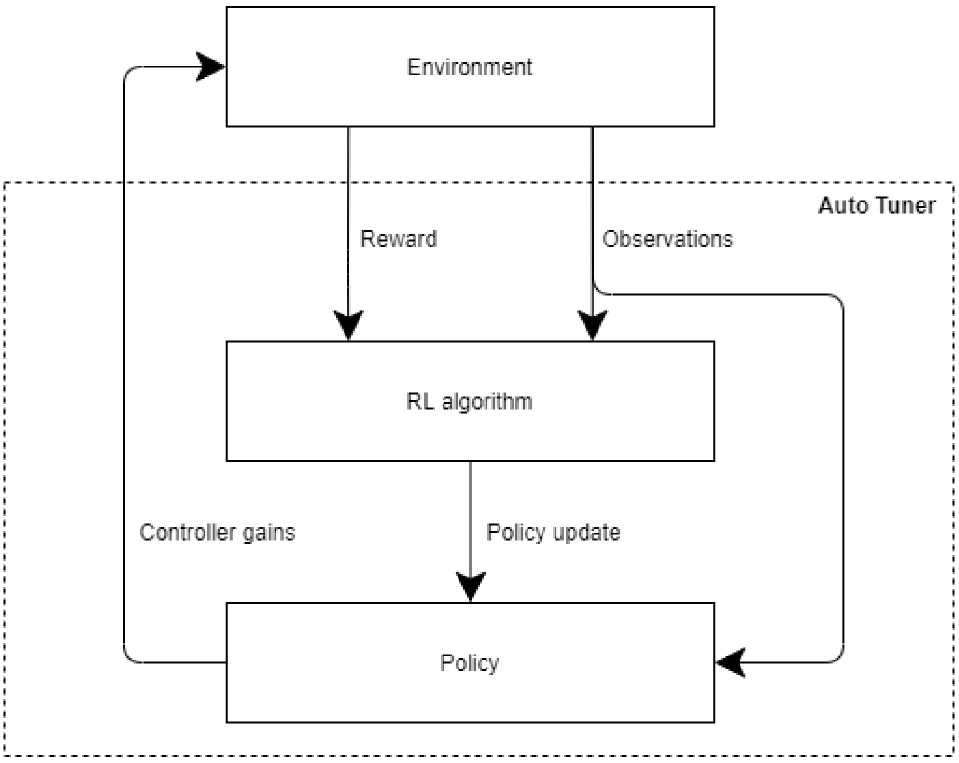

Figure 2: Schematic sketch of the Auto Tuner's structure.

In Table 3, the Auto Tuner's input and output signals are described.

Table 3: Input and output signals of the Auto Tuner.

Data Bus Name	Signal Name	Data Type	Unit	Signal Type
Observations	Position error in x-axis	Float	m	Input
	Position error in y-axis	Float	m	Input
	Heading error	Float	rad	Input
Reward	Reward	Float	-	Input
PID Gains	K_p	Float	-	Output
	K_i	Float	-	Output
	K_d	Float	-	Output
LQ Weight Matrices	Q	Float	-	Output
	R	Float	-	Output

The machine learning algorithms used in the Auto Tuner can be represented as a function $g : O \times R \to P$, where O is the space of possible observation signals o, R is the space of possible reward signals r and P is the space of possible control parameters p. For example could $p \in P$ be a vector containing the controller parameters i.e. $p = [K_p \; K_i \; K_d]$ where K_p, K_i and K_d are conventional real valued PID parameters.

In principle one could define the observation signals to be the current state errors e_t (e.g position error, velocity error and heading error of the AGV) and define the reward signal to be for example the sum of these errors. In this way, the Auto Tuner g would output the parameters p at each time step, thus implicitly forming a control signal

$$u_t = K_p e_t + K_d \frac{\partial e_t}{\partial t} + K_i \int_0^t e_t \, dt. \tag{3}$$

The problem with this approach is that the Auto Tuner itself can be seen as a controller that sends a control signal. In that case, p could as well enter the system after the PID controller as a final touch of the input signal u to the AGV.

To mitigate this problem, we instead propose two changes to the described observation and reward signals. As the first change, the observation and reward signal should be depending on a time interval instead of a single time instance. This could be realized in two different ways. The first way being to define the observation signal at time t as

$$o_t = \{e_t\}_{t=t-T}^{t=t}, \tag{4}$$

where T is some constant. The second way being to define the observation signal as expected value and variance of the observation signal in the first way. As the second change, the Auto Tuner could be defined to only output parameter values at a certain time interval instead of each time step. The two extremes would then be to update the parameters each time step and to only update the parameters once for each simulated scenario.

4.1 Introduction to reinforcement learning

Reinforcement learning can be described as a method to make an agent learn a specific task simply by trial and error without any direct human interaction. This can be done in many different ways, but the core fundamentals are always the same. An Agent interacts with an environment, in our case the environment will be a Simulink model, the agent sends out an action to the environment and takes in the states and a reward that depends on how well the task was solved. The Agent then updates its behavior to maximize its total cumulative reward.

Policy

The policy is a part of the agent and works as a rule book that decides what action a specific state s will result in. The policy is denoted by μ if it's deterministic and π if it is stochastic.

Reward and Value

The reward function r is what that tells the agent how the current policy is performing and it depends on the current states, the actions taken and the next states. To design a good policy it's not enough to only know the reward for a certain action, instead we want to know the total expected reward when starting in a specific state s and following policy π under a time interval. This function is called the value function $V^\pi(s)$.

States and Observations

A state s contains all the relevant information that fully describes the environment in each time step. An observation o is the observable/measurable part of the state. If the state is fully observable, state and observation are the same, otherwise the observation is just a partial description of the state and may omit some information. The agent updates the policy based on the observation and the reward.

Action

The set of all the allowed actions in a specific environment is called action space. It can be either discrete or continuous depending on the environment. The action is a decision made by the agent in order to maximize the future reward.

A short description of all the key concepts in RL is given in Table 4.

Table 4: Key concepts in RL

Concept	Symbol	Description
Agent	-	The controller subject to the learning
Environment	-	The world that the agent lives in and interacts with
Action	a	Control signal(s) decided by the agent
State	s	Complete description of the state of the environment
Reward	r	A numerical value that represents how good the action taken by the agent is
Policy	μ or π	The strategy that the agent follows in order to make decisions
Observation	o	A partial description of a state

4.2 Types of reinforcement learning

There are two main types of reinforcement learning algorithms, model-based and model-free. What type of algorithm that can be used depends on if the agent has access to a full model of the environment. If the agent has access to a full model of the environment, meaning that the value function for starting in all possible states are known, no exploration is needed and a model-free algorithm can be used. In our case a model free method will be used and therefore we need to learn or in some way estimate the reward for a certain action. The two main ways of doing this are policy optimization and Q-learning.

A policy optimization method is usually an on-policy method, which means that each update only uses data collected while acting according to the most recent version of the policy. It is usually denoted as $\pi_\theta(s|a)$ and tries to optimize the parameters θ. It also uses an approximator $V_\phi(s)$ to estimate the value function $V^\pi(s)$ which it uses to update the policy π. Methods based on Q-learning are usually an off-policy method, meaning that each update can use data collected at any point during training, the opposite of the policy optimization. It uses the approximator $Q_\theta(s,a)$ to learn the optimal action-value function $Q^*(s,a)$ and is based on the Bellman equation (see Section 4.4).

4.3 Proximal Policy Optimization

PPO is a policy gradient method that can be used for environments with either discrete or continuous action space [2]. The method is based on optimizing a parametrized policy with respect to the long-term cumulative reward. This project will use a variant of PPO called PPO-Clip. The main idea in PPO-Clip is to avoid large changes between the old policy and the new one when performing a policy update. This is achieved by introducing a specialized clipping in the objective function which prevent the new policy to get too far from the old one. The algorithm is described in detail below.

Let $R_t(\theta)$ denote the ratio between the new policy's and the old policy's PDFs:

$$R_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \tag{5}$$

The objective of the optimization problem that PPO-Clip solves can then be formulated as follows:

$$L_k^{CLIP}(\theta) = \mathbf{E}\left[\sum_{t=1}^{T}\left(min\Big(R_t(\theta)A_t^{\pi_{\theta_k}},\ clip(R_t(\theta),\ 1-\epsilon, 1+\epsilon)A_t^{\pi_{\theta_k}}\Big)\right)\right] \tag{6}$$

where ϵ is a tunable hyperparameter and *clip* is a function that keeps the probability ratio $R_t(\theta)$ within the interval $[1-\epsilon,\ 1+\epsilon]$ by clipping it at the interval's upper and lower bounds. And $A_t^{\pi_{\theta_k}}$ denotes the advantage function corresponding to policy π_{θ_k} and is defined by:

$$A_t^{\pi_{\theta_k}}(s_t, a_t) = Q^{\pi_{\theta_k}}(s_t, a_t) - V_\Phi^{\pi_{\theta_k}}(s_t) \tag{7}$$

where $V_\Phi^{\pi_{\theta_k}}(s_t)$ is the on-policy value function, parametrized by Φ, which gives the expected return if one starts in state s_t and always acts according to policy π_{θ_k}. $Q^{\pi_{\theta_k}}(s_t, a_t)$ is the on-policy action-value function which gives the expected return if one start in state s_t, takes a random action a_t and acts according to policy π_{θ_k} forever after. The problem with this approach is that two value functions are needed which increases the complexity of the algorithm. A common way to overcome this problem is to estimate the advantage function as follows:

$$\hat{A}_t^{\pi_{\theta_k}}(s_t, a_t) = r_t + \gamma V_\Phi^{\pi_{\theta_k}}(s_{t+1}) - V_\Phi^{\pi_{\theta_k}}(s_t) \tag{8}$$

where r_t is the reward in time *t*, and γ is the discount factor which determines how much weight is put on the future and immediate rewards.

Finally, PPO updates the policy by maximizing the objective function, usually by taking minibatches of Stochastic Gradient Descent (SGD) according to:

$$\theta_{k+1} = arg\max_\theta \left(L_k^{CLIP}(\theta)\right) \tag{9}$$

Pseudo-code for the implementation of the PPO-Clip is given below.

Algorithm 1 PPO-Clip [2]

```
Input : Initial policy parameters $\theta_0$ and value function parameters $\Phi_0$
For k = 0,1,2,... do
  Collect set of trajectories $D_k = (\tau_i)$ by running policy $\pi_k = \pi(\theta_k)$
  Compute reward-to-go $\hat{R}_t$
  Compute advantage estimates $\hat{A}_t$
  Update the policy by maximizing the objective function $\theta_{k+1}$ :
      $\theta_{k+1} = arg\max_\theta \frac{1}{|D_k|T}\sum_{\tau\in D_k} L_k^{CLIP}(\theta)$
  Fit the value function $V_\Phi(s)$ by regression on mean squared error:
      $\Phi_{k+1} = arg\min_\Phi \frac{1}{|D_k|T}\sum_{\tau\in D_k}\sum_{t=0}^{T} min\left(V_\Phi(s) - \hat{R}_t\right)$
Endfor
```

4.4 Deep Deterministic Policy Gradient

The DDPG-algorithm is a hybrid of Q-learning and policy gradient and is used for learning in continuous action spaces. The DDPG-algorithm consists of learning two parts which are a Q-function and a policy. The DDPG agent is of the actor-critic type meaning that the algorithm simultaneously learns the policy and value function. The actor, $\mu(s|\theta^{\mu})$, is a policy network that takes in observations and directly returns the action that gives the highest long-term reward, making the policy deterministic. The critic, $Q(s,a|\theta^{Q})$, is a Q-value network that utilizes observations and actions as inputs and returns the expectation of long-term reward (Q-value). To learn the Q-function, it uses off-policy data and the Bellman equation shown in equation (10). Using off-policy data means that computations can be made without considering how the data was generated. The Bellman equation describes an optimal action-value function $Q^*(s,a)$. It is also possible to describe a mean-square error equation of the Bellman equation that computes how well a neural network, $Q_\phi(s,a)$, with network parameters, ϕ, satisfies the Bellman equation as shown in equation (11).

$$Q^*(s,a) = \underset{s'\sim P}{E}\left[r(s,a) + \gamma \underset{a'}{max} Q^*(s',a')\right] \tag{10}$$

$$L(\phi,D) = \underset{(s,a,r,s')\sim D}{E}\left[\Big(Q_\phi(s,a) - \big(r + \gamma \underset{a'}{max} Q_\phi(s',a')\big)\Big)\right] \tag{11}$$

where D is a set of tuples of the state, action, reward and next state (s,a,r,s') and γ is the discount factor that decides the prioritization between immediate and future rewards.

DDPG-algorithm also makes use of target networks that are time-delayed copies of their original networks. If target networks are not used the agent is susceptible to divergence since the network's equations becomes independent on the values calculated by itself, hence the target networks increases stability. Both the actor and the critic each has a target network that works by periodically setting the actor/critic parameters to the latest values which are used to calculate the next state's Q-value. The actor and critic are denoted with Q' and μ' respectively. The target networks has "soft" updates, meaning that only a fraction $(1-\tau)$ of the weights, θ, are transferred, based on main networks as shown in equation (12) and (13).

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)^{Q'} \tag{12}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)^{\mu'} \tag{13}$$

where $(1-\tau) << 1$.

Moreover the training process for the deep neural network can be quite sensitive. This is solved by adding a replay buffer consisting of a set of tuples D. The replay buffer samples experiences in order to update neural network parameters. The size of the buffer affects the algorithms stability. If it is too small the algorithm will overfit to the latest data and if it is too large it will slow down the learning process. The last parts needed to be defined for training the model are the network's updates for the actor and the critic. The Q-values for the next state are computed by a mean square loss function between the original and updated Q-values as shown in equation (14). The purpose of the policy function is to maximize the expected return, J. The expected return is simply calculated by the estimate of

$Q(s,a)$. Maximizing the return is done by taking the derivative of the estimate w.r.t. the policy parameter, θ^{μ}. The fact that a replay buffer is used also needs to be considered which all will result in equation (15).

$$L = \frac{1}{N}\sum_i \Big(y_i - Q(s_i, a_i|\theta^Q)\Big)^2 \tag{14}$$

where y_i is calculated from the Bellman equation.

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \Big(\nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|s_i\Big) \tag{15}$$

In the pseudo-code below the process for the DDPG agent is described.

Algorithm 2 DDPG Algorithm

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^{\mu})$ with weights $\theta^Q$ and $\theta^{\mu}$
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$ , $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer $\mathcal{R}$
for  $For$ episode=1, M  do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  for  $t = 1$, T  do
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action at and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{R}$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{R}$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\Sigma_i(y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
    $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|s_i$
    Update the target networks:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
  end for
end for
```

5 SIMULATOR

Below is a description of the simulator.

5.1 Overview

The simulator consist of the AGV model and the scenario generator. The AGV model is responsible for the simulation of the physical system and to generate system states and measurements of these states given control inputs. The scenario generator provides the system with a path and various disturbances. A blueprint of the simulated system integrated with the rest of the system can be viewed below.

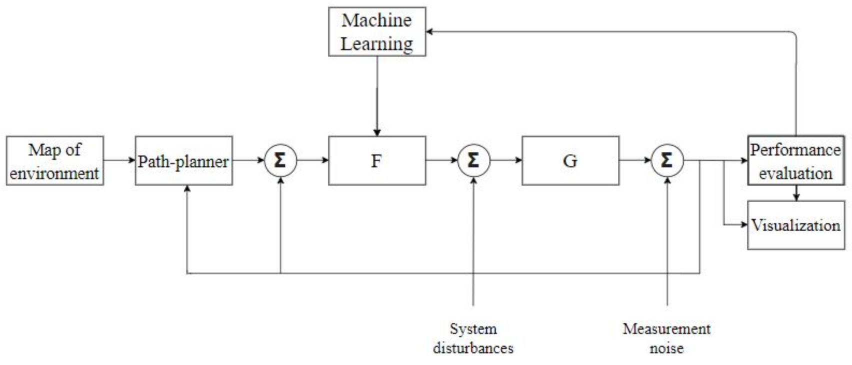

Figure 3: Blueprint for the simulator's interaction with the rest of the system

The simulation process starts with the scenario generator which returns to the simulation with a map of the environment, i.e. what path to take, and with various disturbances. The path is created in the "Map of environment" block and the disturbances in the "System disturbances" and "Measurement noise" signals. A path-planner then provides the controller with a reference signal from the location of the AGV and the environment. This is done in the "Path-planner" block. The controller, the block "F", then controls the AGV model, represented by the block "G", by controlling it towards the reference signal. The performance and observations of its control is later used as an evaluation by the "Performance evaluation" block to tune the controller using ML. The performance can be viewed as a data-file and as a graphical interface.

5.2 User Interface

In this section the user interface of the simulation is described.

5.2.1 *GUI*

Figure 4 shows an outline for the GUI of the simulator.

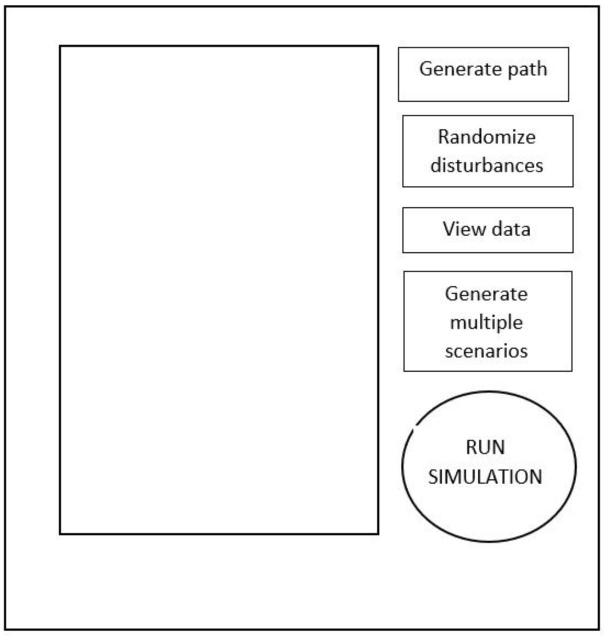

Figure 4: GUI

Generate path allows the user to press the large screen in order to choose locations between which a path will be randomly constructed. The locations are created in Cartesian coordinates, 100x100, and are sent to the Python-script which generates the paths. The environment is free of obstacles as Toyota M.H. only wants us to follow a path. As such, the path can be regarded as a path that avoids obstacles, however these obstacles are non-existant. How the path is created is explained in Section 4.6. *Randomize disturbances* enacts random disturbances and the strength of said disturbances. *View data* opens another window where the user can view important data from the scenario and the simulation. *Generate multiple scenarios* does not display any visualizations but it provides data in the *View data* window.

5.3 AGV Model

The present section describes how the kinematics and dynamics of the AGV will be modelled and implemented in the simulator. The AGV model consists of three main components. The first component is the kinematic model that calculate the position and heading of the AGV from the left and right wheel speed. The second component is the measurement model that adds measurement noise to the position, heading and wheel speed of the AGV. The third component is the dynamical model component that calculates the actual wheel speed of the AGV given the reference wheel speed. An overview of the AGV model is shown in Figure 5.

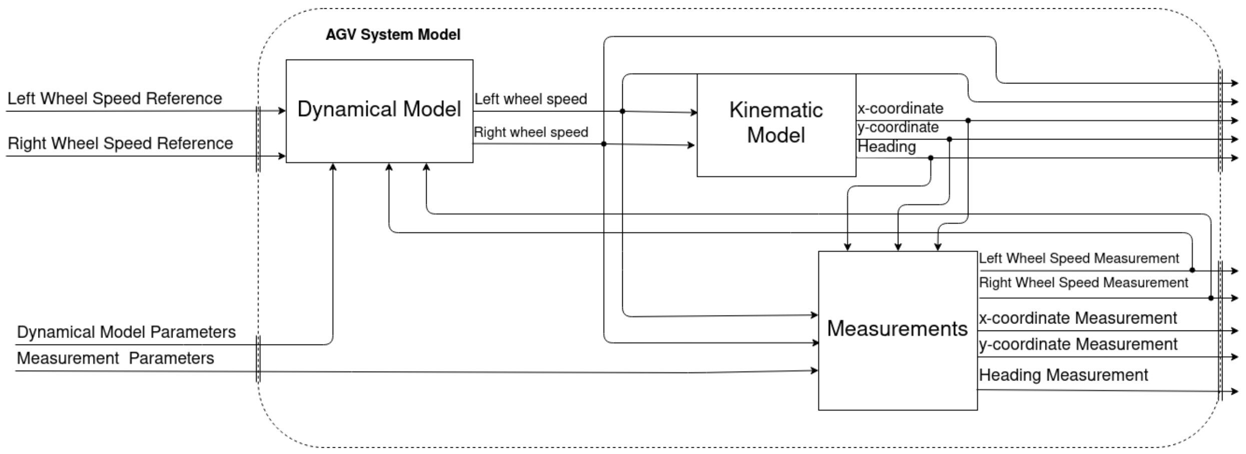

Figure 5: Structure of the AGV System Model

5.3.1 *Interface*

The AGV model component of the simulator takes the control signal sent from the controller as input and outputs the system states and measurements of these states. Table 5 shows the input signals to the AGV model component and Table 6 shows the output signals of the AGV model component.

Table 5: The input signals to the AGV Model component of the simulator.

Data Bus Name	Signal Name	Data Type	Unit
Control Signals	Left Wheel Speed Reference (v_r^{ref})	Float	m/s
	Right Wheel Speed Reference (v_l^{ref})	Float	m/s
Disturbances	Dynamical Model Parameters	Float vector	m/s
	Measurement Parameters	Float vector	m/s

5.3.2 *Kinematic model*

The kinematic model of the AGV is presented below (16-18). The input signals to the model are the velocities of the right wheel (v_r) and the left wheel (v_l). The output signals to the model are the x-position (x), the y-position (y) and the heading (θ).

Table 6: The output signals of the AGV Model component of the simulator.

Data Bus Name	Signal Name	Data Type	Unit
System States	Left Wheel Speed (v_l)	Float	m/s
	Right Wheel Speed (v_r)	Float	m/s
	x-axis Coordinate (x)	Float	m
	y-axis Coordinate (y)	Float	m
	Heading (θ)	Float	rad (Counter clockwise positive)
Measurements	Left Wheel Speed Measurement (v_l^{mes})	Float	m/s
	Right Wheel Speed Measurement (v_r^{mes})	Float	m/s
	x-axis Coordinate Measurement (x^{mes})	Float	m
	y-axis Coordinate Measurement (y^{mes})	Float	m
	Heading Measurement(θ^{mes})	Float	rad

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{16}$$

$$v = \frac{v_r + v_l}{2} \tag{17}$$

$$\omega = \frac{v_r - v_l}{L} \tag{18}$$

where L is the length of the wheel axle, v is the AGV aligned velocity and ω is the angular rate of the vehicle.

5.3.3 *Dynamical Model*

The dynamics model will output the reference wheel speed with an optional time delay. If time allows, this will be extended to model the motors and wheels of the AGV. The dynamical model takes the measured wheel speed as input signal. This is because the real AGV controls each wheel with a PID controller (not to be confused by the controller described in Section 3), that uses the difference between measured wheel speed and reference wheel speed as reference signal [1].

5.3.4 *Measurements*

The measurements are calculated from the simulated system states by adding additive Gaussian noise to the system states (see Table 6). If project resources allow other variations of noise can be tested as well. This could for example be done by estimating the noise of the system states from data of a real AGV that is not moving.

5.3.5 *Validation*

By using the speeds from the log file provided by Toyota M.H. we can measure how much our model outputs differ from the log file. This will allow us to get an understand of how accurate our model is.

5.4 Scenario Generator

The purpose of the scenario generator is to generate a map of the environment and various disturbances that is used by the simulator during simulation and during training of the auto-tuner. Table 7 lists the data contained in each point of the map.

Table 7: The data contained in each trajectory point of the trajectory.

Signal Name	Data Type	Unit
Unique id	Integer	N.A.
Global x-coordinate	Float	m
Global y-coordinate	Float	m
Global Frame Heading	Float	100 000 * rad (Counter Clockwise Positive)
Speed	Float	mm/s

The scenario generator is built on two modules: a path generator and a disturbance generator. The path generator uses RRT (rapidly-exploring random tree) to generate a path between a chosen number of point inside the virtual environment. RRT is a path-planning algorithm. The algorithm can be viewed in Algorithm 3.

Algorithm 3 RRT [5]

```
1:  $\mathcal{T}.init(x_{init})$
2:  for k=1 to K do
3:    $x_{rand} \leftarrow RANDOM\_STATE();$
4:    $x_{near} \leftarrow NEAREST\_NEIGHTBOR(x_{rand}, \mathcal{T});$
5:    $x_{new} \leftarrow STEER(x_{near}, x_{rand}, \Delta x);$
6:    $\mathcal{T}.add\_vertex(x_{new});$
7:    $\mathcal{T}.add\_edge(x_{near}, x_{new});$
8:    if $x_{new} = x_{goal};$
9:      end
10: end for
11: Return $\mathcal{T}$
```

The algorithm works as A random node is selected in the generated environment, which in our case is a 100x100 grid. The algorithm will iterate K times. Afterwards, the nearest neighbor to the random node is selected and the graph is steered towards it; wherever the graph lands is the new node. The two nodes are then connected and, unless the new node is the goal node, the process is repeated. The algorithm either ends when it has iterated K times or when the goal has been reached. The path is constructed of arrays which contain the location id, x- and y-coordinates, the orientation angle, and the velocity. The disturbance generator generates random disturbances: random in the sense that the strength and which disturbances are active varies.

5.4.1 *Feasibility*

The generated scenario needs to be feasible in the sense that the AGV should be capable of executing the scenario without exceeding its physical capabilities. More precisely this means that

- The required acceleration needed for the AGV to move between two consecutive trajectory points should be less than the maximal acceleration of the AGV.
- The required turning rate needed for the AGV to move between two consecutive trajectory points should be less than the maximal turning rate of the AGV.

5.4.2 *Path-planner*

The AGV will have a very basic path-planner; the path-planner will always steer the AGV to the nearest point on the path in relation to the AGV's position.

5.5 Errors

Errors are modeled as white-noise in a vector form allowing us to test the errors independently of each other. Placements of the errors in the model can be seen in Figure 3. A time-delay will be tested by introducing a time-delay block in the simulink model. One before F and one before G in Figure 3

6 EVALUATOR

In this section the evaluator is described. The purpose of the evaluator component is to output the reward signal that is needed to train the auto-tuner (see Section 4.1).

One simple way to calculate a reward signal is as a sum of the closest distance between the reference trajectory and the position of the AGV from the time of the previous action to the time of the current action. This will be used as the baseline reward signal, but other reward functions might be implemented and tested if project resources allow.

6.0.1 *Interface*

Table 8 shows the input to the evaluator component and Table 9 show the output to the evaluator component.

Table 8: The input signals of the evaluator component of the simulator.

Data Bus Name	Signal Name	Data Type	Unit
Path	r	float	-
Measurements	y	float	-
System States	x	float	-

Table 9: The output signals of the evaluator component of the simulator.

Data Bus Name	Signal Name	Data Type	Unit
Reward Signal	Reward Signal	Float	N.A

7 OBSERVATION GENERATOR

In this section the observation generator is described. The purpose of the observation generator is to generate the observation signal to the auto-tuner (see Section 4.1).

In accordance with the discussion in Section 4, the observation signal could for example be sequences of the measurements, control signals and the reference trajectory over a given time. Another alternative could be rolling arithmetic means and variances of those sequences. Experimentation and evaluation of different types of observation signals will be performed during the project within the limitations of project resources.

7.0.1 *Interface*

Table 10 shows the input to the observation generator component and Table 11 shows the output to the observation generator component.

Table 10: The input signals of the observation component of the simulator.

Data Bus Name	Signal Name	Data Type	Unit
Path	r	float	-
Measurements	y	float	-
Control Signal	u	float	-

Table 11: The output signals of the observation component of the simulator.

Data Bus Name	Signal Name	Data Type	Unit
Observation Signal	Observations	float	-

REFERENCES

[1] A. Holgersson and J. Gustafsson, "Trajectory tracking for automated guided vehicle," 2021.

[2] OpenAI, "Proximal policy optimization," 2018, accessed: 2021-10-04. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[4] T. Glad and L. Ljung, *Reglerteori: flervariabla och olinjara metoder*. Studentlitteratur, 2003.

[5] M. LaValle S., "Rapidly-exploring random trees: A new tool for path planning," 1998.

[6] OpenAI, "Key concepts in rl," 2018, accessed: 2021-10-06. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#