Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Mojca Kolšek

Robotski krmilnik za tekmovanje SICK Robot Day 2018

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: dr. Ivan Bratko
Somentor: RNDr. David Obdržálek, Ph.D.

Ljubljana, 2018

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V tem diplomskem delu razvijte krmilni program za robota za tekmovanje SICK Robot Day 2018. V skladu s pravili tekmovanja naj si robot prizadeva zbrati več žog kot nasprotni robot, pri čemer se mora izogibati prepovedanim trkom z nasprotnim robotom. Predpostavite, da se bo vaš krmilni program izvajal v operacijskem sistemu ROS in bo lahko uporabljal pomožne, nižjenivojske funkcije, kot je obdelava osnovnih senzorskih informacij (kamera, laserski senzor) ter osnovno premikanje robota. Zasnova vašega krmilnika naj omogoča čim bolj enostaven vnos izboljšav krmilnika. Krmilnik ovrednotite z izvajanjem na simulatorju.

Faculty of Computer and Information Science issues the thesis:

Topic of the thesis:

The task of this bachelor's thesis project is to develop a robot control program for the SICK Robot Day 2018 Competition. According to the rules of the competition, the robot should try to collect more balls than the opponent robot, whereby avoiding forbidden collisions with the opponent. Assume that your controller will run under the ROS operating system, and that the controller can use auxiliary, lower-level functions, such as basic sensory information processing (camera, laser sensor), and basic robot motion. The design of the controller should enable easy introduction of improvements. Evaluate your controller by its execution on a simulator.

Great thanks to Ivan Bratko, my mentor, for your time and guidance. I sincerely appreciate the help and time of my co-mentor at Charles University of Prague, David Obdržalek. Also, I could not do the project without the help of my team members Alexander Smirnov, Jan Hadrava, Chembrolu Surya Prakash, and Saaram Ali. I would also like to thank my coordinator at Charles University, Kristyna Kysilkova, who did not only guide me through my exchange but also listened to my requests and enabled me to keep the work going. My home faculty coordinators Alenka Kavčič and Ksenija Rozman, and everybody else at the University of Ljubljana and Charles University were of great help as well.

Moji mami, ki je prepoznala moj potencial, in mojemu očetu, ki me je s tehnologijo seznanil že v otroštvu.

And last but never least to all of my friends from Prague and Ljubljana that accompanied me through it all.

Contents

List of Figures

List of used acronyms

Acronim	English	Slovenian
ROS	Robot Operating System	robotski operacijski sistem
RGB	Red Green Blue	rdeča zelena modra
SICK	SICK Sensor Intelligence company	
LiDAR	Light Detection And Ranging	lasersko skeniranje

Povzetek

Naslov: Robotski krmilnik za tekmovanje SICK Robot Day 2018

Avtor: Mojca Kolšek

Podjetje SICK gosti tekmovanje SICK Robot Day v oktobru 2018. Cilj tekmovanja je prenesti žogice iz avtonomnega premikajočega se vozila (imenovanega transporter) v košaro za odlaganje. Hkrati tekmujeta dva robota, ki morata biti povsem avtonomna. Zmaga robot, ki v desetih minutah zbere več žogic.

Moja naloga je bila oblikovati in implementirati robotski krmilnik (tu in naprej imenovan SICKcon) tako, da bo omogočal asinhrono razvijanje neodvisnih delov, kot so razvijanje programa za računalniški vid, razvijanje krmilnika, ki se izogiba trkom, in oblikovanje robotske roke, ki lahko seže po žogice.

Nekateri neodvisni deli, kot so krmilnik za izogibanje trkov in velik del računalniškega vida, niso bili razviti med mojim rokom za oddajo diplomske naloge, zato je okolje v veliki meri simulirano z več programi. Krmilnik zato tudi ni bil testiran v realnem okolju.

Krmilnik sem razvila do stopnje, kjer lahko robot uspešno izvaja pobiranje žogic s transporterja, v kolikor so neodvisni deli implementirani po mojih pričakovanjih. Za oblikovanje robotskega krmilnika sem nalogo razdelila na več podnalog. Za dosego podnalog SICKcon izvaja akcije, kjer so akcije implementirane neodvisno od robotskega krmilnika. SICKcon služi komunikaciji med neodvisnimi deli in odločanju, katera akcija se izvrši kdaj.

SICKcon je implementiran s hierarhičnim avtomatom in končnimi avtomati. Stanja končnih avtomatov služijo kot opis zunanjega sveta, oziroma kaj o zunanjem svetu je znano, ter stopnje v planu.

Evalvacija je na koncu pokazala, da se manjše izboljšave v planu da implementirati hitro in z malo kode, v kolikor so predpostavke o uporabljenih neodvisnih delih uslišane. Za neodvisne dele, za katere prepostavk nisem postavljala, je SICKcon potrebno spremeniti glede na dejansko implementacijo teh delov.

Ključne besede: ROS, robotski krmilnik, končni avtomati, hierarhični avtomati, SICK Robot Day.

English term	**Slovenian term**	**Description**	**Example**
robot controller program	robotski krmilnik	a program that controls the robot's behavior	SICKcon is a robot controller program
SICKcon	SICKcon	the name of my robot controller program	
finite state machine	končni avtomat	used in implementation of SICKcon	SICKcon is implemented with 5 different finite state machines
hierarchical state machine	hierarhični avtomat	used in implementation of SICKcon	SICKcon uses a hierarchical state machine which consists of 4 levels
sensor data	informacija iz senzorjev	data gathered from used sensors	SICKcon makes use of sensor data that comes from camera and LiDAR

Table 1: General terminology

Abstract

Title: Robot Controller Program for the SICK Robot Day 2018 Competition

Author: Mojca Kolšek

The company SICK is hosting a SICK Robot Day 2018 competition in October of 2018. The task of the competition is to make an autonomous robot with the goal of picking up balls from another autonomous, moving robot, and collecting them into a basket.

My task was to design and implement a robot controller program (hereafter referred to as "SICKcon") in a way that allows asynchronous development of independent components such as computer vision, drive controller, and mechanic "hand" design. Unfortunately, some parts were not finished in time and I had to improvise for testing purposes, so a large part is simulated. Evaluation is based completely on simulated environment and the robot has never been tested with real components.

The goal of enabling independent development has not been achieved completely as I have not been able to test how my drive controller would react in real world. However, the goal of adaptability has almost been achieved due to the hierarchy of goals and implementation with many levels which allows partial changes as needed.

Keywords: ROS, robot controller, finite state machine, hierarchical state machines, SICK Robot Day.

Chapter 1

Introduction

The task is to build a fully autonomous robot that is capable of collecting balls in the SICK Robot Day 2018 competition. The task of the thesis is to develop a control program (SICKcon) for the robot. Accessible tools have been acquired and SICKcon makes use of those tools by requesting and receiving the data they can obtain. The task of SICKcon is to determine robot behavior where the actual behavior is implemented independently. SICKcon does not lead the robot's movement nor does it plan a path, however, it plans an appropriate goal point. Collision avoidance is assumed to be handled by another independent component.

SICKcon is a robot controller program that requests independent components to do their part at the right time. If necessary, it passes arguments to them (such as goal point to drive controller). The behavior is fully deterministic and determined by the inner and outer conditions. SICKcon was designed while other team members worked on their tasks, asynchronously. Communication was not our best attribute, thus SICKcon is built with adaptability in mind.

Its implementation is broken down to hierarchy of modules, which can be changed without influencing other modules. Once SICKcon was built, I made quick improvements with very little code. However, when designing SICKcon, I lacked the sensor data, such as camera sensor data, which I had to

simulate and SICKcon was not tested with real components. I assumed what and how I can receive, therefore the sensor data is presented to SICKcon as assumed.

A large part of the competition is collision avoidance which I tried to implement with a simple behavior that SICKcon would use when a danger of collision is present. However, this approach failed to show a decent result. Thus, one level in SICKcon's implementation is dedicated especially for moving the robot but it is also expected to be implemented with a different approach to collision avoidance.

Chapter 2

Rules of the Competition

In this chapter the rules of the competition are presented and the names for the robots in the competition are stated.

2.1 Objects in the Competition

The goal of the competition is to design, implement, and build a robot that picks up balls from an autonomous vehicle and transfers them to a box. The team that collects more balls in 10 minutes wins. Every robot competes in two matches, the better one counts.

Words used for objects in the competition will be:

Rules

The rules refer to the official document, also known as Rules of Procedure, that SICK has published for the competition.

Arena

The Arena is the playing field, marked with stands. It is better described in Section 2.2

Transporter

The transporter is an autonomous vehicle with balls, driving on marked path in the arena. It is better described in Section 2.4.

Robot

The robot refers to the robot of my team, competing in a match.

Storage box

The storage box, also referred to as the basket, is the box in the arena into which the robot has to deliver the balls from the transporter.

Match

A match consists of a competition between two robots and lasts 10 minutes.

Opponent

The opponent robot is the robot of the opposite team.

Refilling stations

The refilling stations are two refilling stations at the sides of the arena.

2.2 Arena

The size of the Arena (Figure 2.1) is 13x7 meters, marked with a fence, 0.5 meter in height. The fence will have advertisements attached. At any given time of a match there are 3 autonomous vehicles in the arena: two competing robots and a transporter. The floor of the arena is a gymnasium floor with additional marking of the path that the transporter follows. At each side of the 7 meter long side of the arena, there is a storage box. Every robot has its own box. At each side of the 13 meter long side of the arena, there is a stand

that marks a refilling station. These marks are 0.5-1 meter high and 0.7-1 meter away from the fence, the length of the marks, however, is not stated. Figure 2.1 is a picture of the Arena as presented in the Rules of Procedure.

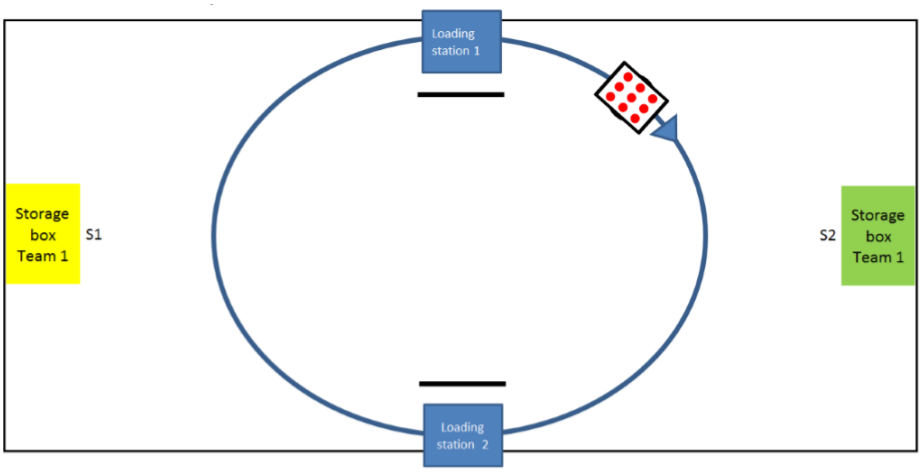

Figure 2.1: The arena

2.3 Restrictions

Disobeying any of the restrictions leads to disqualification. No communication between the team members and their robot is allowed except when stopping the robot in emergency. The emergency button is required. The robot's body width has to be within the 60cm limits. The body of the robot must be at least 25cm high and lifted not more than 15cm from the ground. It can reach further than 60cm (in order to reach the balls on the transporter), but those parts must not be rigid.

Collisions between the competing robots are forbidden. When the robots approach each other from the sides, the right one has the right of way. When approaching directly, the collision is avoided by going right. In case of collision, the responsible robot is disqualified from the match. The responsibility is determined by a judge. A contact with the transporter is allowed (the

transporter starts slowing down and gradually stops when it detects an obstacle), but one point is deducted for every 30 seconds when the transporter is stopped. The transporter must not be moved when making contact.

2.4 Transporter

The transporter follows its path at all times. It has 9 slots for the balls. The balls are red, 6cm in diameter, and weigh 200g each. As mentioned in 2.2, there are two refilling stations at the sides of the arena. The transporter has a constant speed of 0.2m/s, although due to its collision avoidance (already mentioned in 2.3), it can be slowed down and stopped by approaching it from the front. The transporter can be touched but not moved. The Rules of Procedure include pictures (Figure 2.2) with detailed measures of the transporter.

2.5 Not Stated

Although the Rules of Procedure explain a lot about the competition, some information is still missing. The path of the transporter is said to be marked on the floor but nothing is said about its shape. Its shape makes it is possible to calculate the time needed for the transporter to make one cycle. Therefore, the path is taken as an ellipse, based on Figure 2.1 and measurements from Section 2.2. Nothing is stated about the direction of the transporter. From Figure 2.1, the direction is assumed to be clockwise.

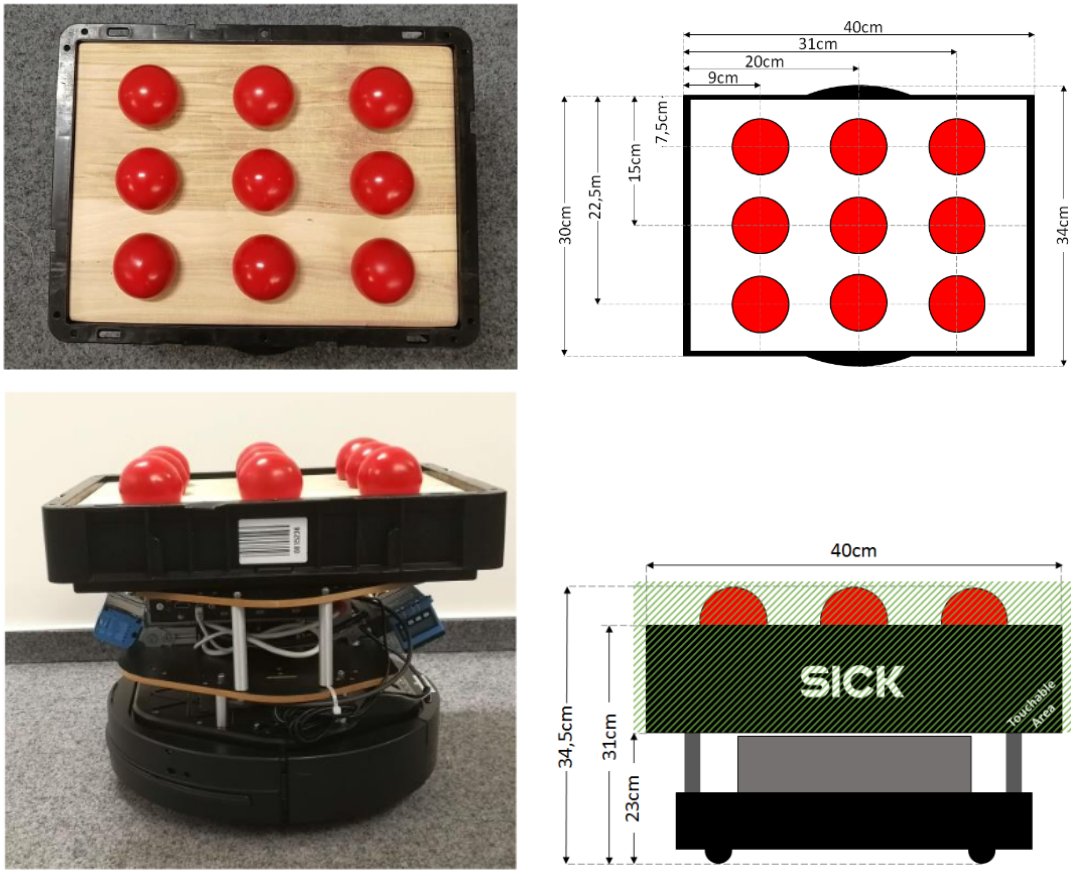

Figure 2.2: The transporter

Chapter 3

Tools Used

This chapter presents the tools used for our task. The goal itself is not to build a robot from start to end, but to build a robot capable of transporting balls as fast as possible. A robot base is used and sensors are attached.

3.1 SICK TIM 571 2D LiDAR Sensor [8]

With a horizontal range of 270° and a distance range up to 25m, LiDAR is the main sensor for collision avoidance.

3.2 Astra Orbbec 3D Camera [1]

With a distance range of 0.6-8m, the robot is able to sense almost half of the arena. It has a 60° horizontal range. With the RGB color sensor, the transporter can be recognized.

3.3 Kobuki Turtlebot 2 [7]

This robot base has a support for developers. Libraries can be used instead of having to program basic functionalities (e.g. differential or twist drive). The robot can handle up to 5kg of payload. Our payload will consist of a

computer, additional sensors, an arm that can reach for the balls, and the balls themselves. The robot base can go up to 0.7m/s, which is 2.5 times faster than the transporter. The battery can hold for much more than 10 minutes which enables us to use its battery for other purposes as well. In addition, this robot has a great support for the Robot Operating System [6] and a Gazebo simulation tool [4, 3].

3.4 Robot Operating System (ROS)

ROS is an operating system that has a good support for the Kobuki robots. The support is wide ranged, containing issue trackers, libraries, discourse forums, and answered questions on other forums.

3.4.1 ROS Node

A node is "a running instance of a ROS program" [10]. At any time, multiple nodes can be running. Any node can be killed by a command line or from a program by calling system calls. When writing a program, a node is initialized and the program can control the node with its handler.

3.4.2 ROS Topic

Topics are a bus system and ROS nodes can communicate with each other by publishing to them and reading from them. Every node can be a subscriber and a publisher to more topics at once. Every topic can have more than one subscriber and/or publisher. A topic is described with a topic message and ROS makes sure that all messages are understood correctly by all languages that have a support for ROS. When a node subscribes to a topic, it is notified when a new message is published and message reception is managed by a callback function, defined when the node subscribes.

3.4.3 ROS Service

Services are another way of communication. They are offered by a node and can be called by any other node. Returning a boolean value, success or failure can be indicated and a return of processed information is possible through a service message request and response. Service messages are programmable, meaning that the creation of the needed service messages is possible.

Chapter 4

Tasks of the Robot Controller

The robot controller is responsible for decision making. The goal is known in advance and SICKcon's responsibility is to connect independent components into a goal oriented, scheduled plan. SICKcon requests execution of behaviors and requests sensor data but the actual behaviors and sensor data gathering is implemented independently.

Assumptions are made to ensure certainty of how the data is passed to SICKcon and what SICKcon can expect when requesting a behavior.

4.1 Assumptions

I assume that the robot is self-localized. The drive controller is an independent component and has a maximum error of 0.1 meters. It is assumed that all sensor data SICKcon requests is accurate, reliable, always available, and returns in a short span of time (less than 1 second). It is assumed that all the behaviors that SICKcon requests are reliable. The transporter's trajectory is an ellipse with known parameters and a center point in the middle of the field. The transporter's direction of travel is known.

However, nothing about the implementation of collision avoidance and pathfinding is assumed. How it could be implemented is further discussed in the next chapter in the topic of how the opponent can interfere with the

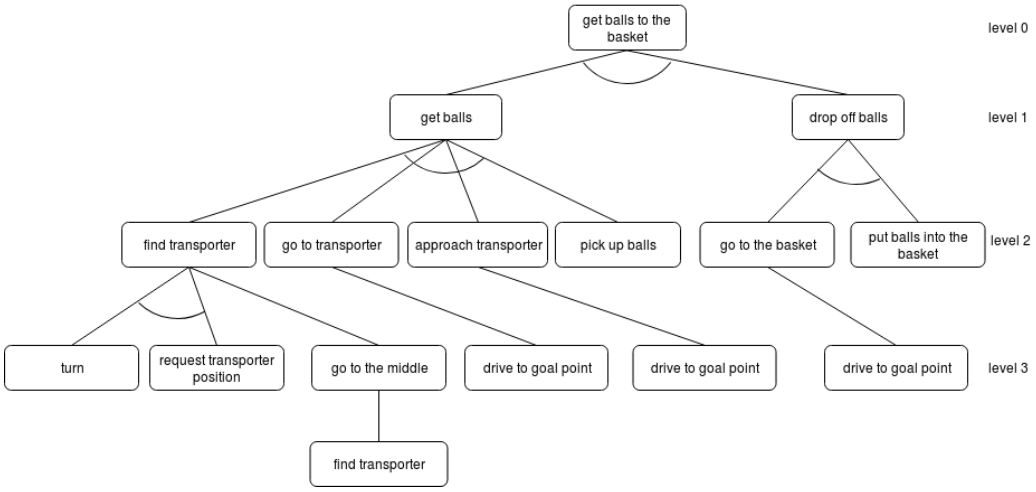

Figure 4.1: Goal tree

movement of my robot.

4.2 Goals

To win, the robot has to get the balls into the basket which is broken down to sub goals as shown in Figure 4.1. SICKcon has to request robot behaviors that reach all the goals in a necessary order, following a scheduled plan. The picture is represented as an AND OR graph and has 5 levels, starting from Level 0. However, the "find transporter" sub goal on Level 5 can be taken as the same "find transporter" sub goal as on Level 2. Therefore, Level 5 can be disregarded as a separate level.

The overall goal (Level 0) is broken down to two sub goals (Level 1). Both sub goals have to be reached in order to get the balls to the basket.

From (including) Level 2 on, the tree is imbalanced, meaning that one sub goal is harder to reach than the other. In order to drop off the balls, there are only two sub goals - driving to the basket and dropping the balls into the basket. Going to the basket is only split down to the drive sub goal.

On the other hand, in order to get the balls, more sub goals have to be

reached. First, we have to find the transporter. When we know where the transporter is, we have to drive to it. The transporter has to be approached from the front to stop it. Once the robot is at the transporter, it can pick the balls up. An alternative would be to approach the transporter from its side and then pick up the balls from it while driving by the transporter's side. For simplicity, however, this approach has been discarded.

Furthermore, the "find transporter" sub goal has been broken down to another level of sub goals. The reason is the camera's range used. The camera's horizontal range is 60°. Thus, the robot has to be turned in the direction of the transporter in order for the camera to see it. The robot has to search the whole area around it to see if the transporter is anywhere around it. The camera's distance range is 8 meters which does not cover the whole arena. Being at the box, a small area can be out of sight. Therefore, the robot has to go to the middle, where it can guarantee the distance to the transporter is less than 8 meters. From there, it can find the transporter by turning for a full circle at its most.

The "drive to goal point" sub goal is reached by avoiding the collision, which is not drawn as a sub goal because SICKcon is not responsible for it.

Chapter 5

Design and Implementation of SICKcon

SICKcon is responsible for reaching all the sub goals (Chapter 4) in the right order. The implementation of SICKcon keeps track of the sub goals that have already been reached at some moment and decides the next necessary action to reach the next sub goal.

The decisions which action to take and which sub goal to reach next are deterministic. The actions that SICKcon takes are requesting sensor data, requesting a behavior, requesting a calculation, and/or passing information on.

If an action is not momentary, meaning it is not expected to be finished in less than a second, SICKcon waits for it to be finished and then takes another action. For example, SICKcon waits for the robot to arrive to a goal point. If an action is momentary, more actions can be executed in a procedural manner, for example, requesting sensor data, requesting a calculation, and then passing a goal point to a drive controller.

SICKcon is built without the consideration of the opponent's behavior but in the next chapter, experiments are shown and some special cases are reconsidered.

I chose to design SICKcon as a combination of the finite state machines

and hierarchical state machines. This approach has already been used in similar robot competitions [9, 12, 11]. I found an open source library written in C++ [2] that allows defining state machines and offers a wide selection of state machine features. From the library I chose the features that allowed me to write SICKcon.

5.1 Library Elements Used

I used only a subset of the elements the library offers. Here, all the used elements are defined and a graphical representation for them is presented. All the names and graphical representations are chosen to be as descriptive and intuitive as possible.

Hierarchical state machine

A hierarchical state machine is defined with a set of finite state machines and parent-child relations. A finite state machine has two parents - a father finite state machine, and a certain state of its parent finite state machine. The finite state machine can only be active if its parents are active, including both its parent finite state machine and its parent state. Thus, only one branch from the root (but not necessarily to the leaf) can be active at any given moment.

Finite state machine

A finite state machine is represented by a rectangle (Figure 5.1a). It is defined with a set of states (Figure 5.1b) and a start state (Figure 5.1c). If a finite state machine is active, exactly one of its states is active at any given moment.

State

A state is defined with a set of actions it performs and is represented by a circle (Figure 5.1b). A state performs actions when it is active, either on

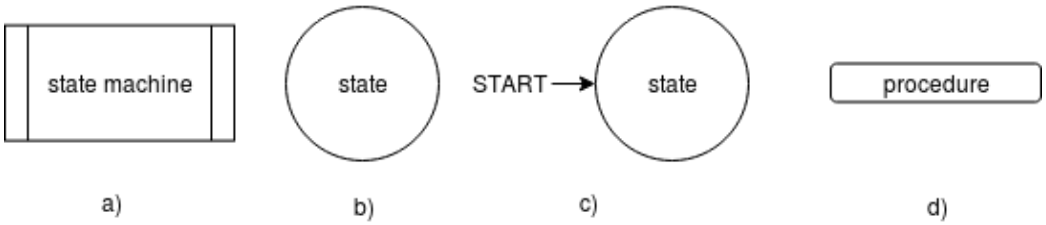

Figure 5.1: Elements

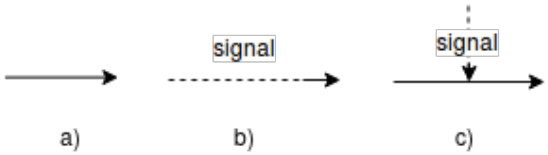

Figure 5.2: Arrows

state activation or as a reaction.

Action

An action is performed by a state. I use three different types of actions: an execution of a procedure, a transition, and a call of a finite state machine. It is represented by an arrow coming from a state and going to an element. An action can be performed on state activation, represented by an arrow (Figure 5.2a), or on a raise of signal, represented by an arrow with an incoming signal (Figure 5.2c).

Signal

A signal is said to be raised and represented by a dotted arrow (Figure 5.2 b). It can be raised by any ROS component. If the component that raises a signal is in the picture, it is drawn by a doted arrow coming from that component. If the signal has no origin, it comes from a component not drawn in the figure

(e.g. sensor). If the signal comes from a finite state machine, it is raised by one of its elements.

Reaction

A reaction is a state performing an action on an incoming signal (Figure 5.2c). A signal is said to trigger an action if the action is defined as a reaction in a currently active state. If a state does not react to a certain signal, a reaction is not drawn.

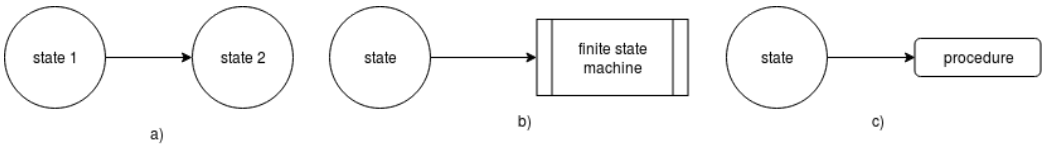

Figure 5.3: A state performing an action

Transition

A transition is a type of action (Figure 5.3a). The state preforming a transition becomes inactive and activates another state. It is always performed as a reaction.

A call of a finite state machine

A call of a finite state machine is a type of action (Figure 5.3b). The calling state stays active. On transition from the calling state, the called finite state machine is stopped. The calling state is named the parent of the finite state machine.

Execution of a procedure

An execution of a procedure is a type of action (Figure 5.3c). A procedure is defined by a C++ code that runs in a new thread and is represented by

a rectangle (Figure 5.1d). It is implemented as an extended ROS "action" component, called a "task" in the library. By the implementation, the calling state does not wait for its execution, but it can raise a signal. Procedures request a robot behavior, sensor data, calculations, and/or pass information to other nodes.

5.2 Structure of SICKcon

SICKcon is built as a single ROS node, communicating with other ROS nodes through ROS services and ROS topics (see Chapter 3). A certain topic is created by the library used. The library interprets every message that is published on this topic as a signal raise. If any of the currently active states react to the raised signal, the library executes the specified action. By publishing on this topic, any node can raise a signal. Furthermore, SICKcon is built of the library elements defined above. In this section, all of the elements will intertwine with one another - procedures, states, finite state machines, signals, and robot behavior.

SICKcon is built with levels to allow necessary changes if the assumptions cannot be met or to enable simple improvements in the plan that SICKcon follows. This chapter represents the version of SICKcon that is able to collect the balls in the basket. In the next chapter, parts of SICKcon are changed to improve the overall result.

5.2.1 The Hierarchical State Machine

SICKcon consists of finite state machines ordered in 4 levels (starting with 0), represented in Figure 5.4. All finite state machines represent one hierarchical state machine. It is structured in the way that allows changes without influencing the lower levels.

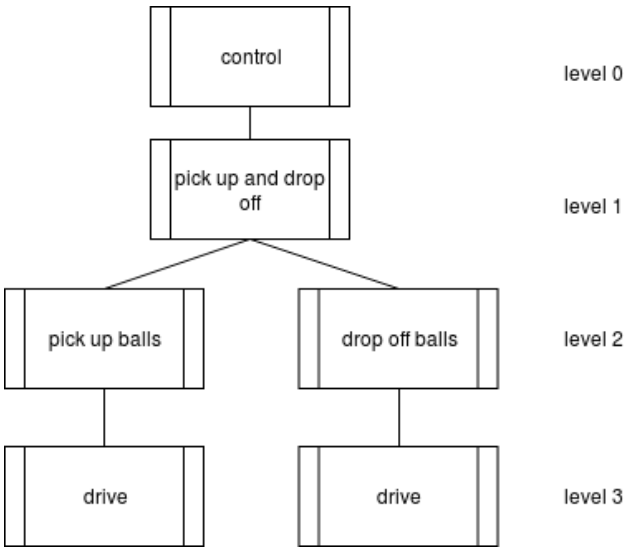

Figure 5.4: The hierarchical state machine

Levels 0 and 1 are separated due to very different responsibilities. The responsibilities that Level 0 carries wait for the "start" signal of a match and listen to the "emergency stop" signal. The responsibility of Level 1 is to keep track of two sub goals.

Contrarily, Level 2 is separated from Level 1 due to breaking down goals to their sub goals, matching the sub goal tree from Chapter 4. In the next section, the resemblance between the sub goal tree branching and the complexity of the finite state machines is observed.

Finally, Level 3 is separated from the others for a different reason. The "drive" finite state machine is much more connected to the pathfinding and collision avoidance approaches than to robot behaviors. This level serves as a communication between SICKcon and a drive controller, depending on how much communication is needed. Although Figure 5.4 shows it only reuses one finite state machine, regardless of its father state machine, this representation only serves for representing hierarchy of the finite state machines. Depending on the implementation of a drive controller, pathfinding and collision avoidance, there can be more different finite state machines on this level. These finite state machines are expected to be designed and implemented along

with the drive controller, collision avoidance and pathfinding.

5.2.2 Finite State Machines

The finite state machines are the brain of SICKcon. Every state represents a certain internal and/or external state, e.g. the position of the transporter (relative to the robot), knowledge about the transporter's position, stage in the scheduled plan, and more.

All the finite state machines are represented by symbols described in Section 5.1 and match the source code. However, there are two exceptions.

The "drive" finite state machine (and none of its elements) does not raise the "at goal point" signal. This signal, in my case, is raised by the drive controller, which I do not focus on. However, it is represented that the finite state machine raises the signal for easier perception of the reader. This signal is always raised by a component that has awareness of the robot's position.

The "pick up" and "drop off" procedures only raise the "picked up" and "dropped off" signals for the testing purposes. They should be raised by a component that controls the movement and the "pick up" and "drop off" procedures only request behavior.

5.2.3 Level 0 - "Control" Finite State Machine

Level 0 (Figure 5.5) is the control state machine, which is the root of the tree and is activated on SICKcon's node initialization. It starts with the "init" state and waits for the "start" signal which triggers the transition to the "start" state. This state calls the Level 1 "pick up and drop off" finite state machine (Figure 5.6). Before the "start" signal, other nodes can still run, e.g. nodes that control the camera and LiDAR.

The "start" state reacts on the "emergency stop" signal by calling the "stop" procedure which disables all moving parts (the drive controller and/or movement of the robot's "hand") and shuts down all running nodes. It also stops the whole "control" finite state machine which stops all the other active

finite state machines.

By making this the root of the SICKcon hierarchical state machine, an emergency stop is possible at any given moment from the "start" signal on.

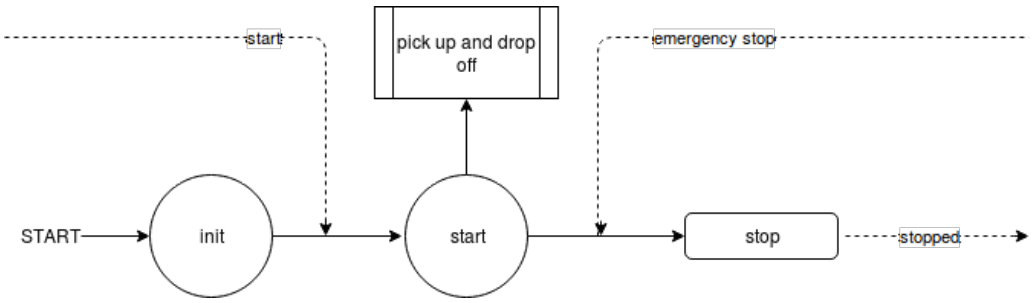

Figure 5.5: Finite state machine, Level 0, "control"

5.2.4 Level 1 - "Pick Up and Drop Off" State Machine

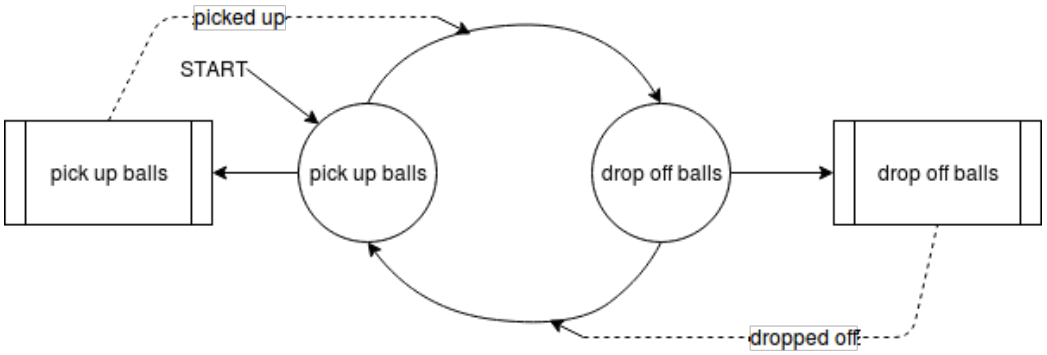

Figure 5.6: Finite state machine, Level 1, "pick up and drop off"

This state machine represents the matching Level 1 in the goal tree (Figure 4.1) and it only has two states matching the sub goals. At the start of a match, the robot has no balls and first has to retrieve them from the

transporter, thus the start state of this state machine is the "pick up balls" state. After the goal is reached (indicated by the "picked up" signal), the state transitions into the "drop off balls" state. This finite state machine can only be stopped by the parent finite state machine. Here, the plan is: repeat "pick up balls" and "drop off balls" (Figure 5.6).

Both of these goals are broken down to a different set of sub goals (see Figure 4.1), thus both states call different finite state machines. Those two finite state machines (rather one of their states) raise the signals that their parent states react to with a transition to the other state.

5.2.5 Level 2 - "Pick Up Balls" Finite State Machine and "Drop off Balls" Finite State Machine

At this level, goals are not broken down to sub goals any more. This indicates the need for procedures through which SICKcon requests sensor data, passes information to other components, requests behaviors and/or requests calculations. According to the sub goal tree (Figure 4.1), the "pick up balls" state machine (5.7) is much more complicated compared to the "drop off balls" state machine (Figure 5.8).

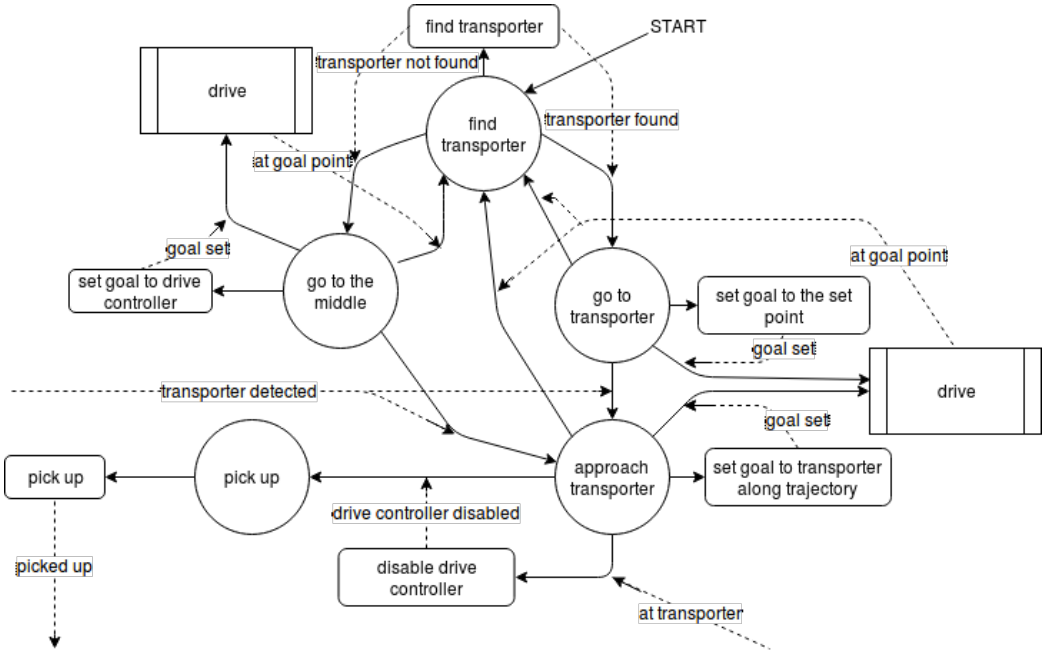

Figure 5.7: Finite state machine, Level 2, "pick up balls"

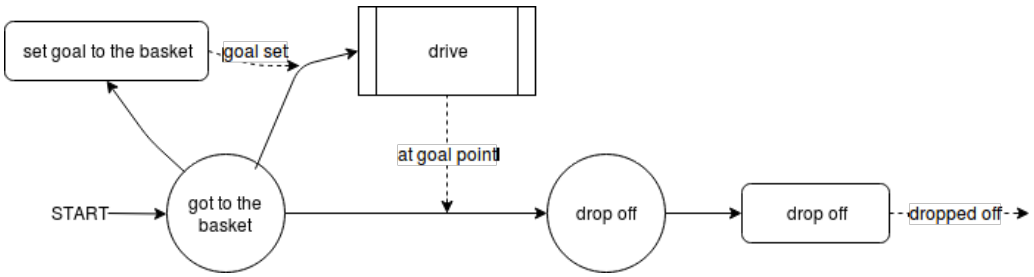

Figure 5.8: Finite state machine, Level 2, "drop off balls"

"Pick up balls" finite state machine

The first sub goal is to find the transporter, so the robot can approach it. As described in Chapter 4, the robot has to be turned in the right direction and

be close enough to the transporter for the camera to see it. This includes turning the robot and going to the middle of the field if necessary.

The starting state performs the "find transporter" procedure, turns the robot and requests the camera to pass the coordinates of the transporter (through a service offered by another node - the one that controls the camera). The camera either passes the coordinates or the "false" value, indicating the transporter is not in the camera's range. If the camera cannot see the transporter, the procedure turns the robot for about 40° (e.g. through a service offered by another node) and requests the coordinates again. If at one of the iterations the camera responds with the transporter's coordinates, the procedure raises the "transporter found" signal, to which the "find transporter" state reacts with a transition to the "go to transporter" state. If after about 10 iterations (when the robot has turned for 360° or more) the camera still cannot see the transporter, it must be too far away for the camera to see it. Correspondingly, this procedure raises the "transporter not found" signal, to which the calling state reacts with a transition to the "go to the middle" state.

The "go to the middle" state performs the "set goal to the middle" procedure, which communicates with the node responsible for moving the robot, passing a desired goal point. After the "at goal point" signal is raised, the state transitions back to the "find transporter" state.

Once the transporter is found and its coordinates are obtained from the camera, the "go to transporter" state is activated. Going directly to those coordinates would not be the best idea since the transporter is moving. It would most likely result in the robot missing the transporter, meaning that the transporter would be further down along its trajectory. Instead, I tried to calculate an appropriate meeting point. The meeting point should be further down the transporter's trajectory, in its direction of travel.

The meeting point is calculated by another independent component that has the information about the parameters of the transporter's trajectory. The "set goal to the set point" procedure requests the component to calculate

and return a set point based on the robot's position and the transporter's position. The calculated set point is further passed on to an appropriate component as the goal point.

When the "at goal point" signal is raised, the state machine goes back to the start state where it repeats the previous steps. This time the camera should see the transporter in one of the "find transporter" procedure iterations.

If, while trying to meet the transporter at the meeting point (the "go to transporter" state is active), the "transporter detected" signal is raised (by a camera or LiDAR), a transition to "approach transporter" is performed. The "go to the middle" state also reacts to that signal with a transition to the same state in case the camera/LiDAR spots the transporter while the robot is on its way to the middle of the field.

The "transporter detected" signal should be raised when the transporter is about 2 meters or less away from the robot. The "set goal to transporter along trajectory" procedure, which is called by the "approach transporter" state on its activation, has to calculate a new goal of travel. It sets the goal to the transporter coordinates and also sets a desired path that leads to that goal along the trajectory.

If the robot happens to reach the goal point without getting the "at transporter" signal, an unexpected set of events has happened (e.g. the path has been prolonged due to collision avoidance). The "at goal point" signal thus triggers the transition to the "find transporter" state and the previous steps are repeated. Otherwise, the "at transporter" signal has to be raised (by camera/LiDAR) indicating we are at the appropriate distance from the transporter to pick up the balls. The state reacts by stopping the robot's movement and transitioning to the "pick up" state. This state performs the "pick up" procedure which requests the mechanic "hand" to reach for the balls. The "picked up" signal triggers a transition in the Level 1 finite state machine and thus stops the "pick up balls" finite state machine.

"Drop off balls" finite state machine

This state machine is far simpler which matches the goal tree (Figure 4.1).

It starts in the "go to the basket" state which performs the "set goal to the basket" procedure. This procedure passes a point at the basket to an appropriate component as the goal point. When the robot has reached the basket and the "at goal point" signal has been raised, the "drop off" procedure is performed. Once the balls are dropped off in the basket, the "dropped off" signal is raised, which triggers a transition in the Level 1 finite state machine and thus stops the "drop off balls" finite state machine.

5.2.6 Level 4 - "Drive" Finite State Machine

This level is different from the others. It is closely connected with a drive controller implementation and should therefore be implemented accordingly. SICKcon's job is to plan a goal rather than to plan a path. Pathfinding can be a part of a drive controller or another independent component. Collision avoidance can be a part of pathfinding and must not be too complex communication-wise due to moving obstacles. This level serves as a communication between SICKcon and a drive controller.

For testing purposes, I have written a simple drive controller that only receives a goal and can be enabled and disabled - all by calling ROS services. The finite state machine represented here is implemented according to my drive controller implementation.

Example of implementation

When the drive controller is enabled, it turns the robot in the direction of the goal point and raises the "turned" signal, which triggers a transition to the "drive" state. This state reacts to the "opponent detected" signal by calling the "avoid opponent" procedure. The procedure requests the point to be avoided and passes it to the drive controller through a ROS service implemented for this purpose. The drive controller is responsible for calcu-

lating the path around that point. When the robot reaches the goal point, the drive controller raises the "at goal point" signal. This signal always triggers an action in the parent finite state of this state machine, therefore the finite state machine stops.

The "turning to goal point" state does not react to the "opponent detected" signal since the robot cannot cause a collision while turning. Turning to the goal point also shortens the path and with it the time of travel to the goal point.

Different finite state machines for different cases

There is another reason for this level of the hierarchical state machine. In different cases, the robot should react differently when approaching either the transporter or the opponent. This level can therefore include more than one finite state machine, with the same purpose and slight differences (Figure 5.9). The "drive" state machine is called by 4 different states: the "go to the middle" state, "go to transporter" state, "approach transporter" state and "go to the basket" state. While the "go to the middle" and the "go to transporter" states react to the "transporter detected" signal with a transition to the "approach transporter" state, the robot should actually avoid the transporter when driving to the basket (Figure 5.9a). And while the robot is far away from the transporter and approaching it, it should not avoid the transporter, it should avoid the opponent (Figure 5.9b). However, it might be better for it to stop only and wait so the opponent moves if the robot is already very close to the transporter (Figure 5.9c). Nevertheless, stopping the robot can lead to endless waiting if the opponent also stops at this point.

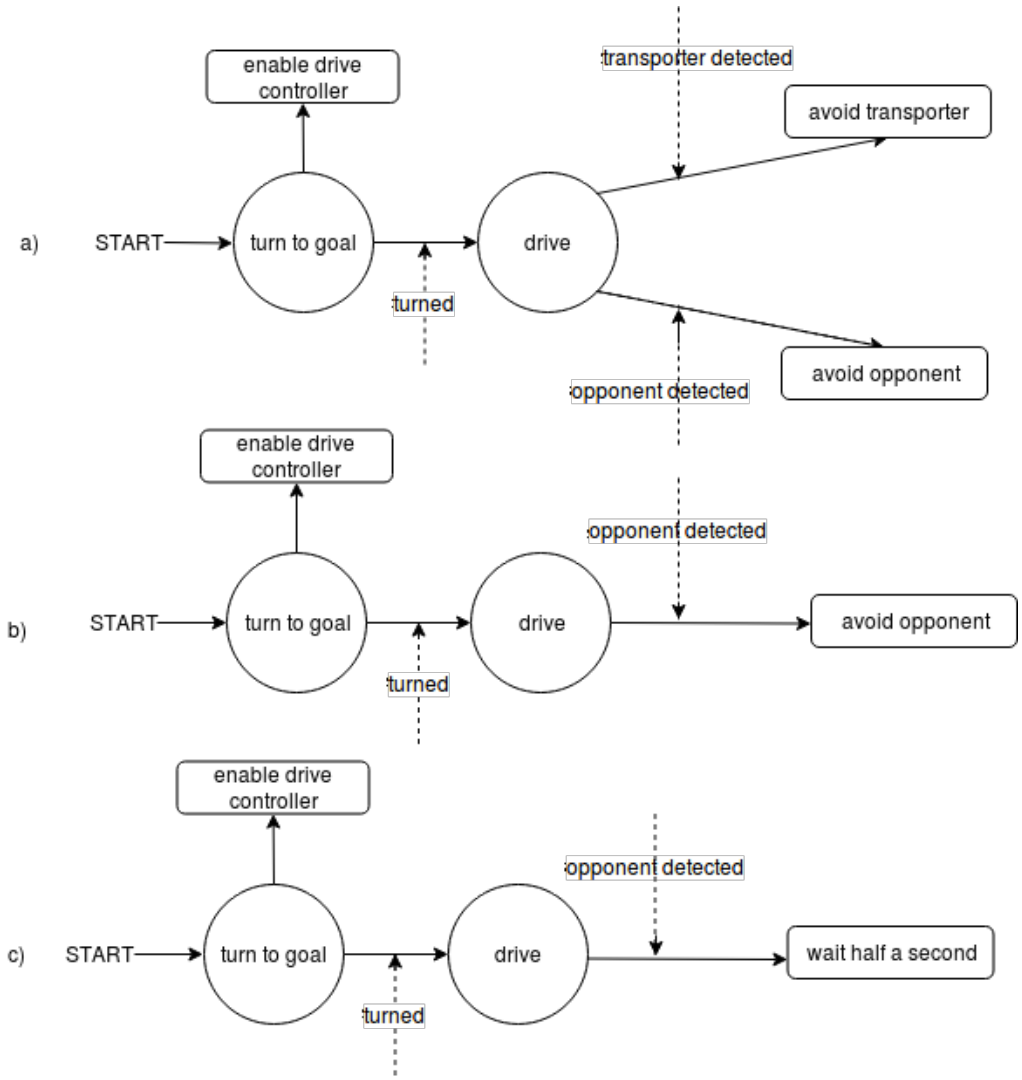

Figure 5.9: Finite state machines, Level 3, "drive"

This is only an idea of an implementation. It disregards the option that the opponent might try to block the robot's way on purpose. A more complex solution would be appropriate. The solution should plan a path tactically which would lead by the opponent's right side, gaining the right of way in the process. Pathfinding and collision avoidance should work together as one component instead of my implementation where collision avoidance has its

own logic. As this is an important part of the match, but no pathfinding approaches have been chosen, I tried to keep this level as open to changes as possible.

5.3 Signals Used

Most signals are raised by procedures, indicating their success. Those signals have to be raised at the end of each procedure. However, some signals are raised by other nodes, such as the node that controls the camera or LiDAR. These have to be raised in the conditions that SICKcon expects.

"Start" signal

This signal should be raised by pressing a button. The match starts with a sound signal and then the team is allowed to press a button on their robot to start its movement.

"Emergency stop" signal

An emergency stop is required by the rules and this is the only case after the start of the match when communication between a team and their robot is allowed. The emergency stop is also raised by pressing a button.

"Transporter detected" signal

SICKcon expects this signal when the camera can see the transporter even while the robot is moving and is able to give the transporter's coordinates accurately. Also, the transporter should be relatively close to the robot - at the distance of about 2 meters. At this point, the robot is more likely to approach the transporter without being distracted by collision avoidance. It can be published more than once, it can even be published with a certain period, but the publication time depends on the ability of the computer that runs SICKcon and all the other nodes.

"At transporter" signal

This signal should be raised when the robot is at a very short distance from the transporter. The proper distance depends on how far the mechanic "hand" can reach. The robot should also be in front of the transporter so that it can be stopped.

"Turned" signal

This signal has to be raised by the node that is asked to turn the robot. The turning action can last more than one second. Services are intended for quick processes (time span of less than a second). Thus, a service is used which requests the turning of the robot, while its success is indicated by a signal from another component.

"At goal point" signal

This signal has to be raised by a node that serves as a drive controller. The reason and purpose is the same as for the "turned" signal.

"Opponent detected" signal

This signal is mentioned for the purposes of collision avoidance. I implemented it to demonstrate an example of the "drive" state machine. In this case, it should be published in the same way as the "transporter detected" signal.

Chapter 6

Evaluation and Improvements

Evaluation is based on the Gazebo robot simulation [4]. The simulation supports the use of Kobuki Turtlebot 2, so the robot is of the same size as the actual robot that has been acquired for the competition. The robot has a maximum speed of 0.7m/s and a maximum turning speed of 180deg/s. It also supports a simulated use of cameras and LiDAR which helps with testing other components. Its odometry data is precise as long as the robots do not collide and will be used for self-localization.

For testing purposes, the sensor data, which is supposed to come from the camera, is simulated. All the signals that should have been raised according to the sensor data, are simulated by a program and based on what the actual sensors would be able to see and are raised as described in (previous) Chapter 5. The service is programmed which returns the "true" value and the transporter's coordinates if the transporter is within 8-meter distance range and 60° horizontal range, otherwise it returns the "false" value. The camera is assumed to be at the center of the robot, facing the same direction as the robot and placed higher than the opponent so it can see past the possible obstacles. I attached lines to the robot that represent the range of the camera for the viewer's easier perception. I marked the robots with different color balls floating above them to help keep track of them. The transporter is marked by a red ball and the two opposing robots with a green and a blue

ball.

The trajectory of the transporter is a circle with a diameter of 6 meters with the center in the middle of the field. These parameters show the cases where the transporter is not in the camera's range. The transporter's way of travel is clockwise. Its speed is 0.2m/s as stated in the rules. The trajectory is not drawn. None of the 9 balls on the transporter are drawn.

The arena is built in the simulation as presented in Figure 6.1. The arena borders are drawn by black lines. The boxes are drawn by two boxes, marked with numbers 1 and 2. The box marked with 1 belongs to the blue robot, the one marked with 2 belongs to the green robot. The refilling station borders are not drawn. Due to the lack of pathfinding and only elementary implementation of collision avoidance, none of these objects are marked as collide objects in the simulation, meaning a robot cannot collide with them.

All the mentioned videos of simulation are uploaded to my github page [5].

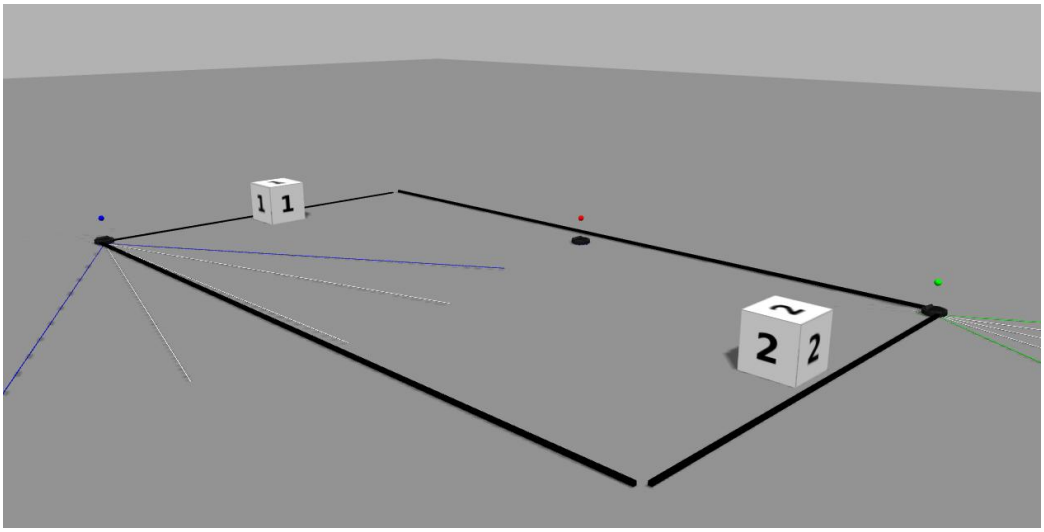

Figure 6.1: Simulation

Because the "hand" is not simulated, the "pick up" and "drop off" procedures raise the "picked up" or "dropped off" signals after 3 seconds, while the robot is not moving.

Based on the simulation, some improvements are being made. The improvements are minor changes to the plan that SICKcon follows and are made by changing the SICKcon finite state machines. All the changes to the state machines are represented by new pictures. What has been deleted is drawn in red. What has been added is drawn in green. For a better view, new states are enlarged. What has not been changed is drawn in black.

Sketches are included where the behavior of the robot is presented. The circle is the robot and the square on the drawn trajectory is the transporter, both presented with their positions in time. The starting position is represented by number 1. The squares represent the states that SICKcon is in and also describe the robot's behavior, and the arrows represent transitions to another state with a signal that has triggered the transitions. Only the states of the "pick up balls" and "drop off balls" finite state machines are drawn.

6.1 Testing Without Opponent

First, I tested SICKcon without an opponent. The robot behaves as expected, presented by Figure 6.2 and by Video 1. It starts on the signal and reaches the "pick up balls" finite state machine, starting in the "find transporter" state. It makes a full 360° turn while searching for the robot. The robot is not in the camera's range, therefore the robot travels into the middle of the field. It starts turning again, searching for the transporter. This time the transporter is in the distance range of the camera and SICKcon gets the coordinates. The "go to transporter" procedure calculates an appropriate meeting point and the robot travels to it. The transporter stops and waits in place because it detects an obstacle while the robot starts finding the transporter again. It finds it after almost a full turn and when the "pick

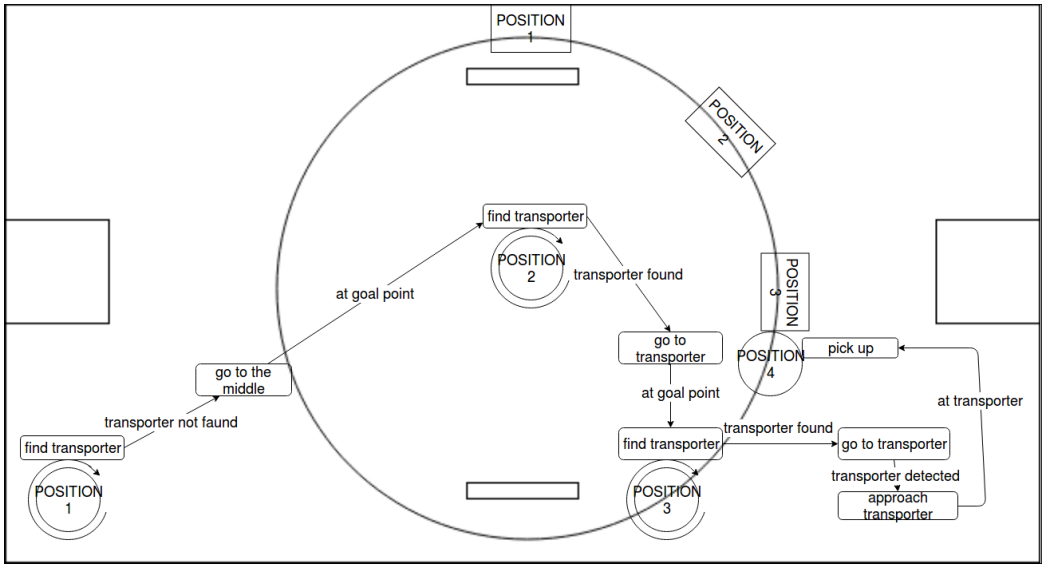

Figure 6.2: Robot behavior - "pick up balls"

up balls" finite state machine transitions to the "go to transporter" state, it immediately reacts to the "transporter detected" signal and transitions to the "approach transporter" state. It successfully approaches the transporter.

After the "picked up" signal is raised, the robot successfully reaches its "drop off balls" state machine. The "drop off balls" state machine leads the robot to the box and drops the balls off as in Figure 6.3. The whole process is repeated.

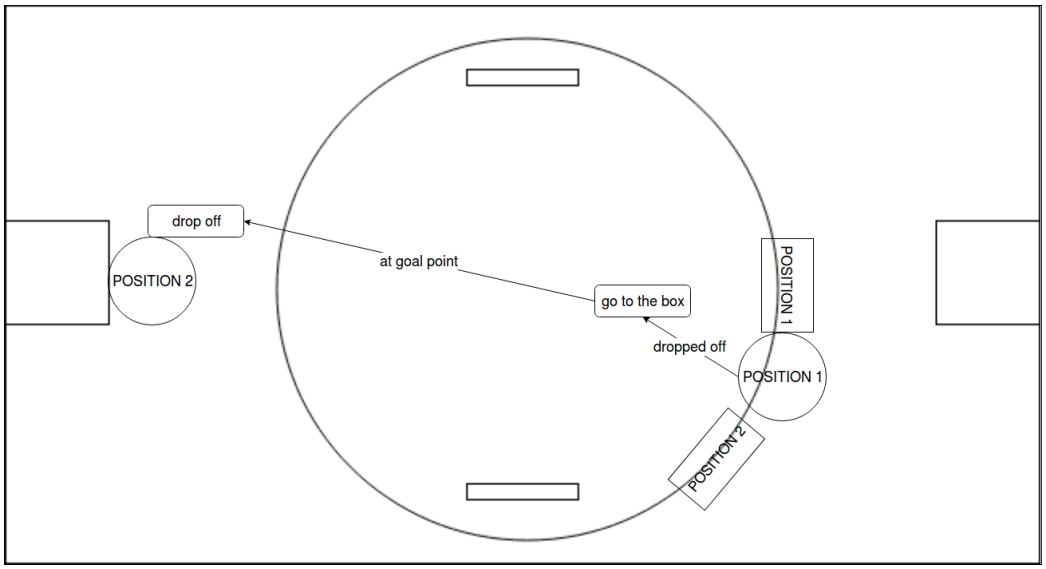

Figure 6.3: Robot behavior - "drop off balls"

6.2 Improvements

After watching the behavior of the robot, it becomes clear there are options for improvements. Improvements are implemented by changing the finite state machines. They add states and shift, add and/or remove some of the signals. They do not affect the states or the finite state machines that have not been changed.

6.2.1 Turn to Transporter

The first improvement I thought of is the stage when the robot transitions from the "go to transporter" to the "find transporter" state. It starts turning clockwise and eventually finds the transporter. However, if we look in the direction where we expect the transporter to be, we would find it much faster. For this purpose I introduced another state in the "pick up balls" finite state machine.

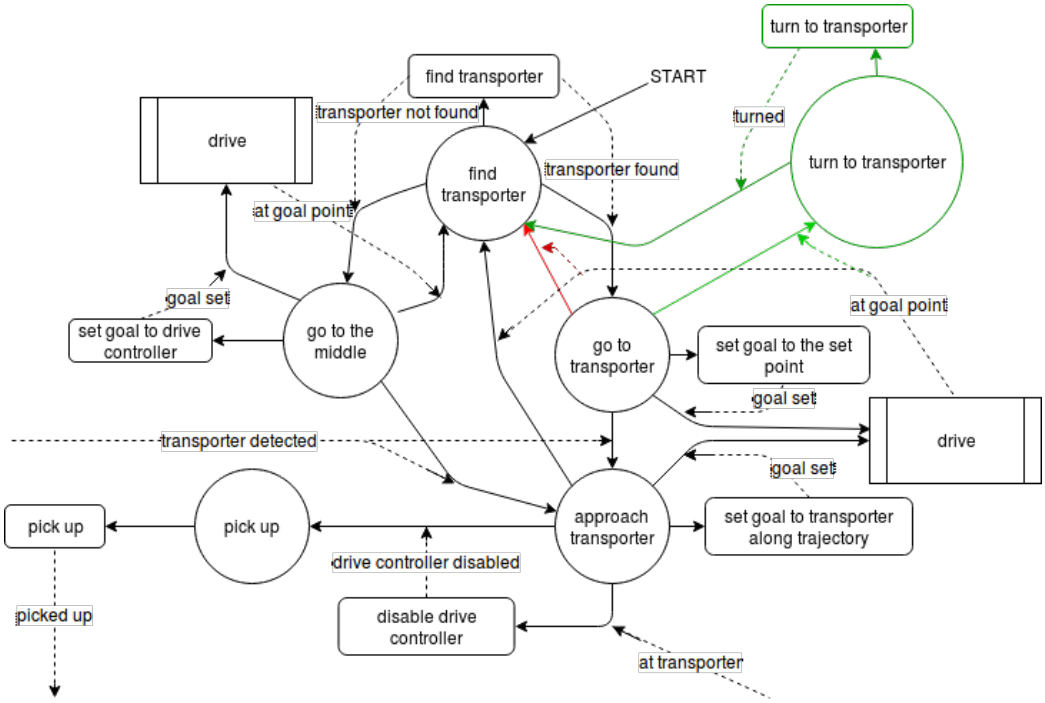

Figure 6.4: Finite state machine, Level 2, "pick up balls" - Improvement 1

Video 2 shows how this affects the robot's behavior. After the robot arrives at the transporter's trajectory, it turns in the direction where the transporter is expected to be. It transitions to the "find transporter" state, where it immediately finds it and transitions to the "go to transporter" state. As before the adjustment, this state reacts to the "transporter detected" signal and transitions to the "approach transporter" state.

Adding one state shortens the time of finding the transporter as well as the time of the transporter being stopped.

6.2.2 Finding the Transporter from the Box

The second possible improvement is changing the start state of the "pick up balls" state machine. From the box, the transporter will be found between the points of the two refilling stations. From the middle, the transporter will be found anywhere in the 360° radius. The "pick up balls" state machine is only called when the robot is at the box or at the start of the match, thus only the start state of this finite state machine has to be changed.

The "find transporter from the box" procedure first requests a turn of the robot to the refilling station on the left of the arena (relative to the robot's drop off box), and then turns for about 180°. This again shortens the time of finding the transporter. Video 3 on my github page [5] shows this improvement paired with the previous one.

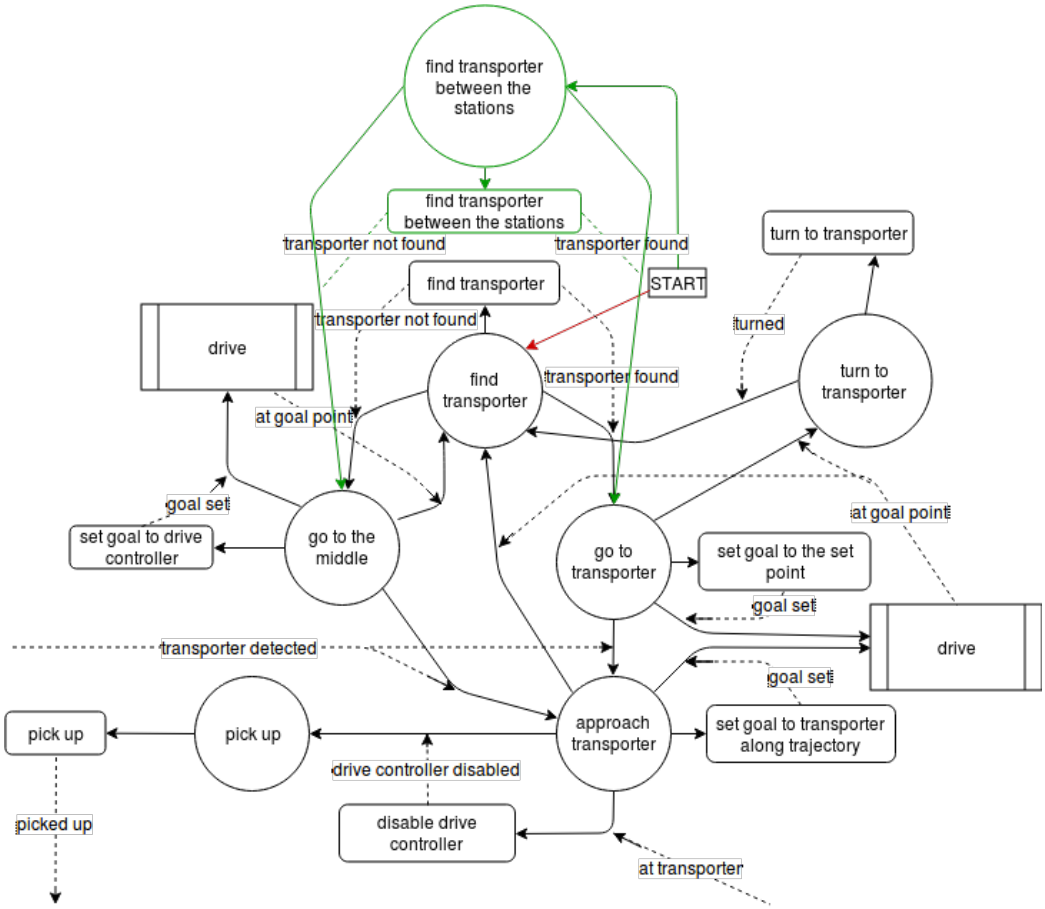

Figure 6.5: Finite state machine, Level 2, "pick up balls" - Improvement 2

6.3 The Opponent's Interference

Because a collision with the opponent leads to disqualification, the algorithm for collision avoidance should be implemented with a high degree of certainty. This can lead to a large path length increase and bad predictions of the transporter's position. SICKcon is built with that in mind and keeps searching for the transporter and trying to confirm if it really is where it is expected to be.

The opponent's interference must be considered. I tried to determine the strategies the opponent might have for winning the game and then determine how my robot could react to them.

Two more problems are special cases of the opponent's interference: first, when the opponent is where the robot wants to be and second, when the opponent is at the transporter, already picking up the balls off it. Those two cases are considered at Level 2 in the SICKcon hierarchical machine while the pathfinding and collision avoidance are a part of Level 3.

6.3.1 Opponent Blocks the Robot

This approach can win the opponent a victory if my robot causes a collision. If the opponent's approach is well implemented and my robot does not cause a collision, the match can result in a draw.

The problem

The opponent might block my robot on purpose. This can be done by following the transporter in the way that enables my robot to approach it. Another way is following my robot and trying to block its way. If it tried to block my robot by stopping the transporter and not move away, it is punished with negative points for every 30 seconds when the transporter is blocked, meaning this approach can be discarded.

The solution

The key is not to cause a collision. Therefore, every path has to be chosen in the way that gains the right of way.

The rules state that, in case when the opposing robots approach each other, the one on the right has the right of way (Section 2.3). A robot can therefore gain the right of way by planning a path that goes by the right side of the opponent as represented by Figure 6.6.

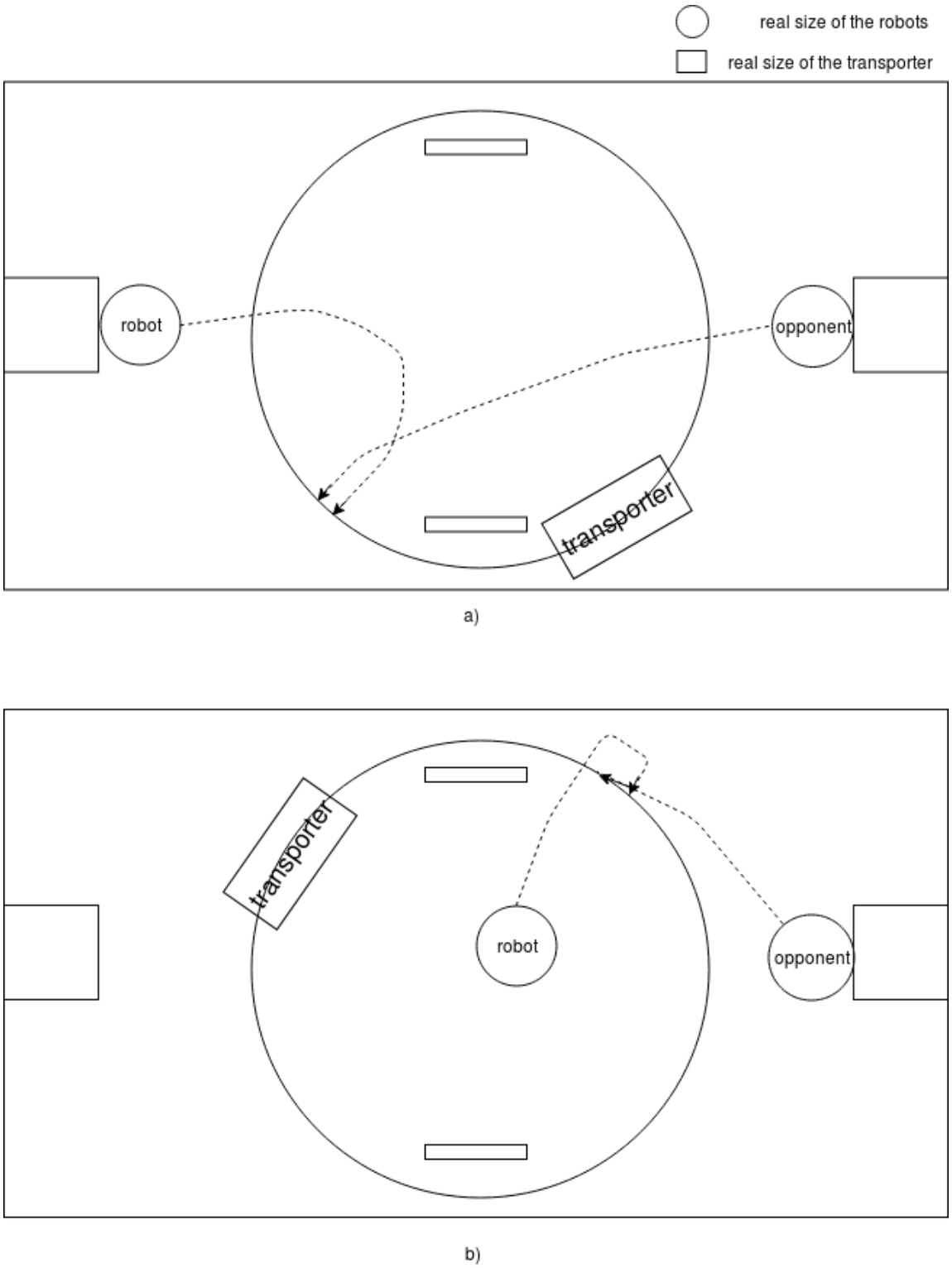

Figure 6.6: Path with the right of way

The other part is to avoid the collision when the opponent is in a critical zone, e.g. two meters in a 60° radius in front of the robot. The collision is avoided to the right.

The success of this solution is highly dependent on how well collision avoidance is implemented, how good a pathfinding algorithm is, and how well does the opponent block the robot's way.

6.3.2 Opponent is Collecting the Balls

This approach is the one I took and it only involves picking the balls up and dropping them off without trying to distract the other robot.

The problem

The opponent can either try to directly approach the transporter by following its changing position or try to meet it at an appropriate point (the way my robot does). The path calculated to meet the transporter is either the shortest path or the path that gains the right of way.

The solution

The path my robot follows should gain the right of way if the time my robot needs to approach the transporter is close to the time needed by the opponent. Otherwise, my robot approaches the transporter (along the fastest path) faster than the opponent and the collision does not happen. Collision avoidance is still necessary and should avoid the opponent by going to the right but only when it is really necessary to avoid the opponent, e.g. when travelling to the basket. If the opponent is already at the transporter (when my robot wants to approach it) it should wait for its turn.

6.3.3 Opponent Uses the Combination of Collecting the Balls and Blocking the Robot

In this approach, the opponent can either try to distract my robot before or after it approaches the transporter, or both.

The problem

The robot can try to approach my robot instead of approaching the transporter, block my robot's way, and approach the transporter when it can approach it before my robot can.

After my robot or the opponent has the balls, it can block my robot's way not to make it to the drop off basket or the transporter. The opponent can then decide to drop off the balls when it is really close to its drop off basket. In this way it can distract my robot from picking up the balls for the maximum of the time needed for the transporter to make one circle.

The solution

This solution is a combination of the solutions from Sections 6.3.1 and 6.3.2.

The path my robot follows should gain the right of way if the time that my robot needs to approach the transporter is close to the time that the opponent needs.

The other part is to avoid the collision when the opponent is in a critical zone, e.g. two meters in a 60° radius in front of the robot. The collision is avoided to the right.

6.3.4 Using the Appropriate Solution

To use an appropriate technique, the opponent's approach has to be recognized first. Of course, the pathfinding algorithm is not trivial when searching for a path that gains the right of way. Machine learning can be applied in both cases.

Determining the opponent's approach

An online machine learning can be applied as a technique to recognize the opponent's approach to avoiding the collision. The strategies described in 6.3.1, 6.3.2, and 6.3.3 can be taken as classes and machine learning classifies the opponent's strategy and applies an appropriate technique to plan the path and avoid collision.

For gathering the training and test data, many simulations have to be made, appropriate attributes selected and a lot of programming done for the robots to move in different ways and follow different strategies.

A path with the right of way

The calculation of a path that would gain the right of way needs a prediction of a path (or at least a goal point) that the opponent travels to. The prediction itself is not an easy task but could be done in example by simulating several matches with different opponents and somehow apply the results. The use of "different opponents" here refers to the opponents that act differently when choosing their blocking techniques.

The results can be applied as training and test data for a machine learning algorithm. A prediction itself can therefore become a complex part of pathfinding. The results can also be applied as a research of how the opponent could try to interfere with the robot on purpose.

Machine learning is just an example of a possible approach. Another approach is using the techniques for two-player games, taking into consideration that the game is continuous.

6.3.5 Opponent at the Robot's Goal Point

There are two static points that SICKcon uses - the point at the drop off box and the point in the middle of the field. Both points have appropriate substitutions; the box has three empty sides where a point at any side of the box is an appropriate goal point and the middle of the field can be slightly

moved.

However, the solution still has to keep the independence of Levels 3 and 4 of the hierarchical state machine. The goal is set at Level 3, thus a change of goal should be handled at the same level.

Another procedure is added to the "set goal to the middle" (Figure 6.7a) and "set goal to the basket" (Figure 6.7b) states, which changes the goal point. The signal that the states react to can be either "path not found" or "goal point taken" or even both, with the same reaction.

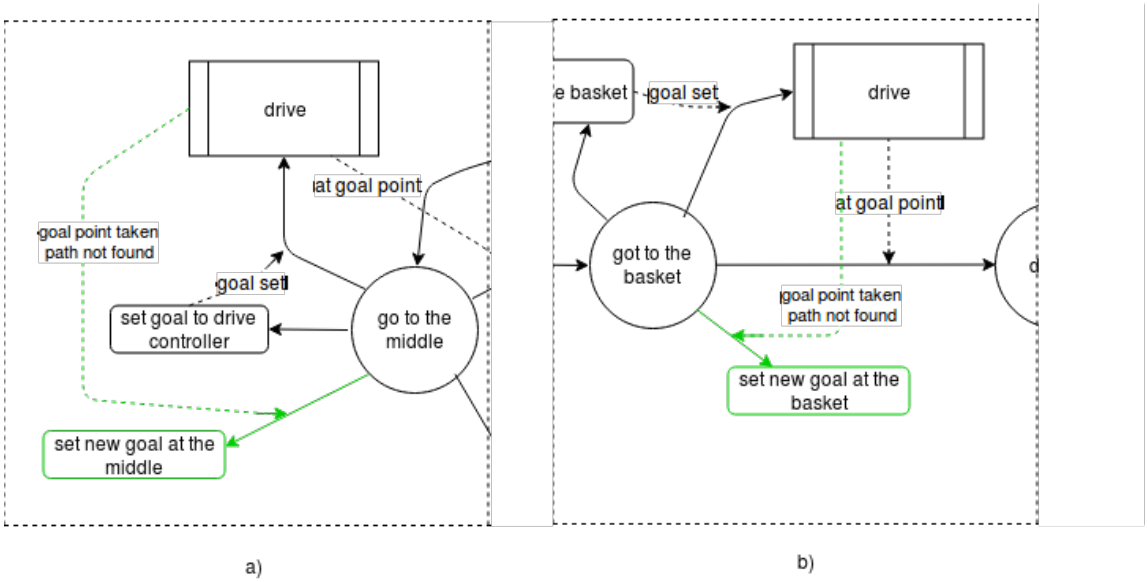

Figure 6.7: Changing the goal point

6.3.6 Opponent at the Transporter

If the opponent is already at the transporter when the robot is approaching it, the robot has to wait for its turn. Determining a new goal is not an option since another appropriate goal might not exist. A better option is to wait for the path or the goal point to clear.

With my implementation of the drive controller, SICKcon avoids the opponent when receiving the "opponent detected" signal. The robot behavior is represented by Figure 6.8, where the opponent and the transporter do not move and the robot moves as shown with the positions. Videos 4 and 5, uploaded to my github page ([5]), represent this scenario. When it successfully avoids the opponent, it approaches the transporter, but it approaches it from the side. Moreover, the opponent is still at the transporter and most likely reaching for the balls, thus the robot can cause a collision.

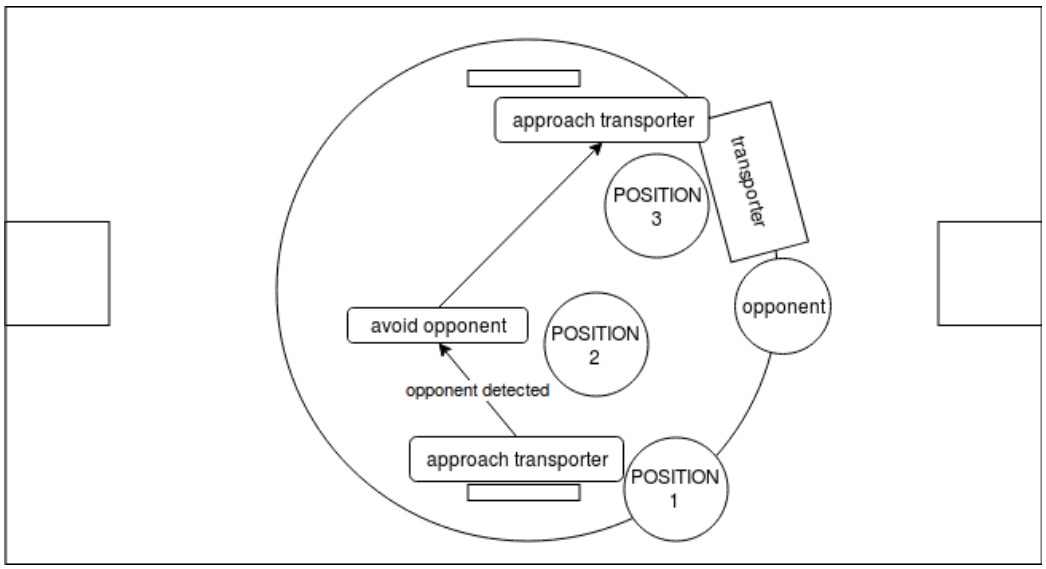

Figure 6.8: Opponent at transporter

A solution to this problem can depend on the drive controller. The "approach transporter" and "go to transporter" states do not react to the "path not found" signal. For this solution, the drive controller must not try to move the robot if a path is not found.

The second solution is to add a reaction to the "approach transporter" and "go to transporter" states that disables the controller for a short time when the "opponent detected" signal is raised. If the signal is raised periodically,

with rate r, the drive controller should be disabled for the time 1/r. When the opponent moves away, the signal stops raising and the robot can continue with its approach to the transporter. This solution would be appropriate for my implementation of the drive controller.

6.4 The Transporter Without Balls

If the camera can see the transporter and also deliver the information about the number of balls on it, another improvement is possible. There are two refilling stations (see Chapter 3) where all free ball slots on the transporter are reloaded. The idea is not to approach the transporter if it does not have the balls on it, but wait for it behind one of the refilling stations.

Another similar case scenario is to have a mechanical "hand" that is able to pick up all of the 9 balls on the transporter at once and another box at the robot that serves as a storage box. The robot can therefore go to the transporter, pick up all the balls on it and wait for the transporter behind the next refilling station instead of dropping them off immediately in the drop off box.

6.4.1 Extended Goal Tree

The overall process of fulfilling the goal tree changes as shown in Figure 6.9. The "get balls" and "drop off balls" sub goals remain unchanged while another sub goal, "get more balls" is added on Level 1. This sub goal is broken down to its own sub goals.

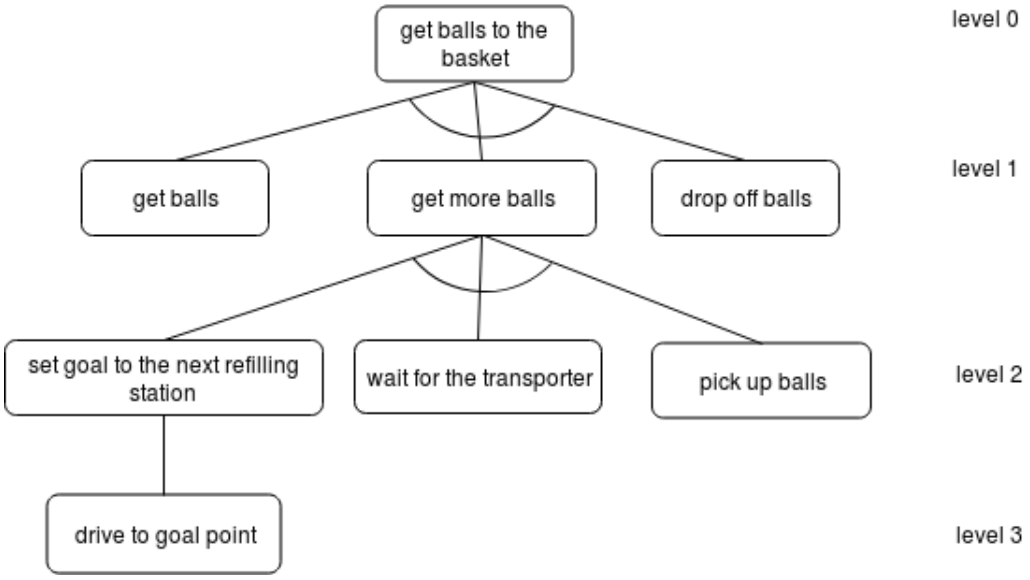

Figure 6.9: Extended goal tree

6.4.2 Implementation

As with the "get balls" and "drop off balls", the new sub goal on Level 1 gets a state in the Level 1 finite state machine and its own finite state machine on Level 2. The new hierarchical state machine is represented in Figure 6.10.

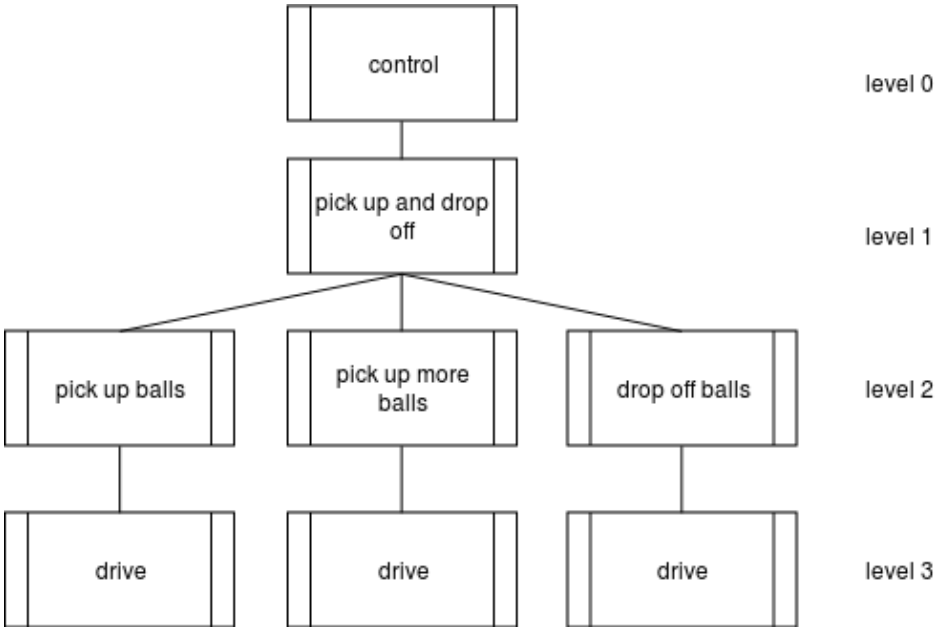

Figure 6.10: Extended hierarchical state machine

Extended Level 1

I added one more state to Level 1 of the hierarchical state machine. The red parts of Figure 6.11 are deleted and green parts are added. In addition, the same transition can be triggered by different signals.

The transition from the "pick up balls" state to the "pick up more balls" state can be triggered by two different signals. The "picked up" signal triggers the transition with an assumption that the robot has a storage box attached to itself. Therefore, more balls can be accumulated in the box before dropping them off to the drop off box. The "transporter empty" signal indicates that the transporter was found, but it has no balls on it.

The transition from the "pick up more balls" state to the "drop off balls" state can be triggered by three different signals. The transition with the "picked up" signal is only valid if the robot does not have a storage box and the previous transition is only triggered by the "transporter empty" signal.

In the case of the storage box, this transition has to be triggered with the "box full" signal. Lastly, in the case of the storage box, the transition should be triggered when the time of the match is running out; the robot should not wait for another set of balls, it should drop them off immediately.

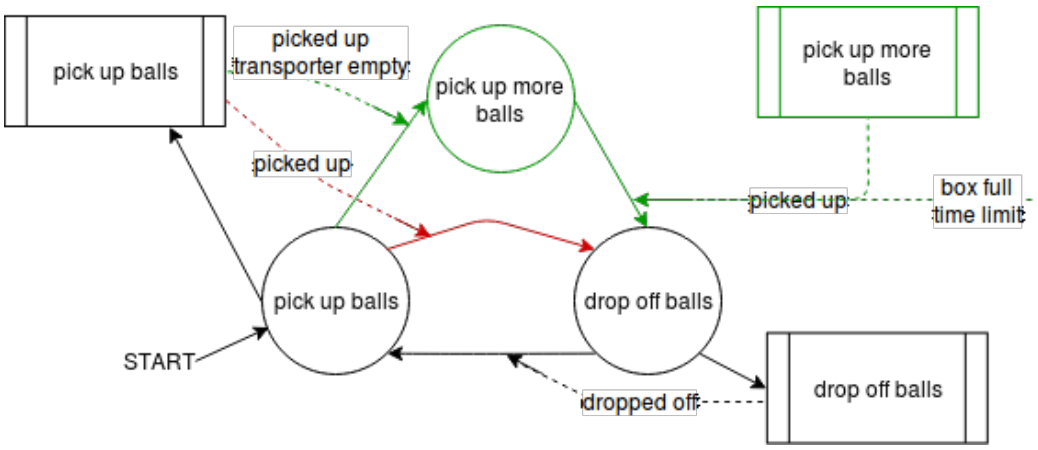

Figure 6.11: Extended "pick up and drop off" finite state machine

Extended Level 2

The new finite state machine is represented in Figure 6.12. It starts with the "set goal to refilling station" state and calls the procedure with the same name. According to the finite state machine on Level 1, the camera has already seen the transporter and SICKcon knows its position. If the camera passes information that the transporter has no balls on it, the camera has seen the transporter and passed its coordinates to SICKcon. If the transition was triggered by the "picked up" signal, the robot has just been at the transporter, therefore, SICKcon knows where it is.

The "set goal to refilling station" procedure simply passes the goal point at the next refilling station in the transporter's direction of travel.

When the robot is at the goal point, it has to turn to the refilling station to appropriately stop the transporter and wait for the "at transporter" signal.

This signal triggers a transition to the "pick up" state which calls the "pick up" procedure.

All the signals have already been used and there are only two new procedures to be programmed. With a small change on every level, a big strategy improvement has been designed and implemented.

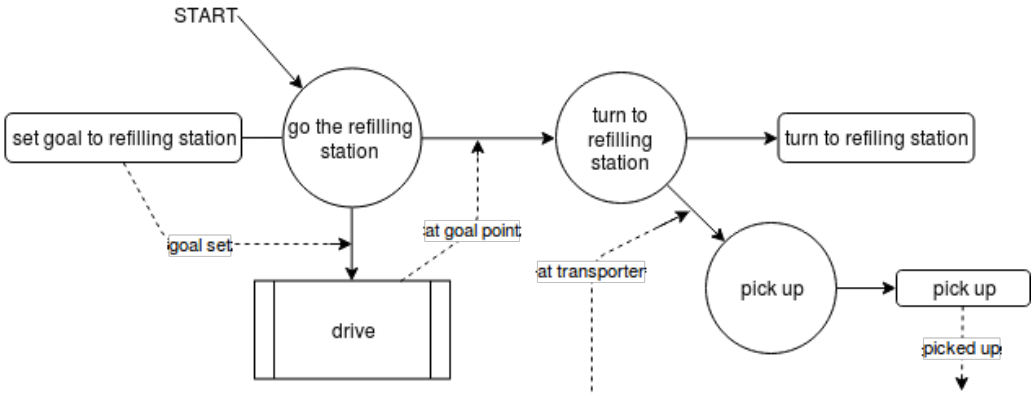

Figure 6.12: Finite state machine, Level 2, "pick up more balls"

Chapter 7

Conclusion

SICKcon is divided into levels according to their purposes and a level of abstraction which allows a certain degree adaptability.

The requirements are collected with the assumptions I have made and signals I have defined. If the requirements are met, SICKcon can fulfil the task and minor improvements can be made easily and with little code.

An important part of SICKcon is how the components that SICKcon uses are implemented. Any improvements to independent components that implement the behavior, collecting and processing data, algorithm for pathfinding, collision avoidance, design of the mechanical "hand" and so on, improve the overall result.

The independent components can be implemented independently, but they have to meet the requirements set by SICKcon. If any of the requirements cannot be met, SICKcon's layered architecture allows changing the levels on which the requirements are not met without influencing other levels. In the example when the camera is substituted for another model that has a better range of the one expected, the behavior can stay the same. Contrarily, if the camera has s shorter or narrower range, a part of SICKcon has to be altered. In this particular case, however, only the procedures that make use of the camera have to be changed and the architecture can stay unchanged.

Nevertheless, Level 3 of SICKcon hierarchical state machine is better left unimplemented. Any implementation that is not in accordance with the implementation of moving the robot to a goal point has to be changed completely. This offers a high degree of freedom for an implementation but also requires the whole level of SICKcon hierarchical state machine to be implemented along with it.

Only after the last level has been implemented, a true evaluation of how well SICKcon can lead the robot can be tested. Unfortunately, I failed to implement it well enough to test SICKcon with an opponent. A significant part of SICKcon is thus left out and cannot be implemented independently.

In this project, the goal of enabling asynchronous development was partly reached as an important part of SICKcon cannot be implemented without a side by side implementation of what should be an independent component. The goal of easy adaptability was partly reached as either the requirements have to be met or a big part of SICKcon has to be changed in case the requirements are not strong enough or not met.

Bibliography

[1] Astra orbbec 3d camera. https://orbbec3d.com/product-astra/. Accessed: May 27, 2018.

[2] cogniteam decision making. http://wiki.ros.org/decision_making. Accessed: July 16, 2018.

[3] Gazebo ros. http://wiki.ros.org/gazebo. Accessed: July 7, 2018.

[4] Gazebo simulator. http://gazebosim.org/. Accessed: July 7, 2018.

[5] Github page. https://github.com/mk6614/SICKcon. Accessed: August 29, 2018.

[6] Kobuki ros. http://wiki.ros.org/Robots/TurtleBot. Accessed: May 27, 2018.

[7] Kobuki yujin base. http://kobuki.yujinrobot.com/about2/. Accessed: May 27, 2018.

[8] SICK tim 571 2d lidar. https://www.sick.com/de/en/detection-and-ranging-solutions/2d-lidar-sensors/tim5xx/tim571-2050101/p/p412444. Accessed: May 27, 2018.

[9] Martin Lötzsch, Joscha Bach, Hans-Dieter Burkhard, and Matthias Jüngel. Designing agent behavior with the extensible agent behavior specification language xabsl. In *Robot Soccer World Cup*, pages 114–124. Springer, 2003.

[10] Jason M. O'Kane. *A Gentle Introduction to ROS*. October 2013. Available at `http://www.cse.sc.edu/~jokane/agitr/`.

[11] Max Risler and Oskar von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in xabsl. In *AAMAS08-workshop on formal models and methods for multi-robot systems, Estoril, Portugal*. Citeseer, 2008.

[12] JM Yánez, L Leottau, P Cano, M Mattamala, W Celedón, M Silva, C Silva, P Saavedra, P Miranda, Y Tsutsumi, et al. Uchile robotics team team description paper robocup 2013-standard platform league.