

Name: _____

Class: _____

Date: _____

Task 1

Mecanum vehicle with obstacle detection

Construction task

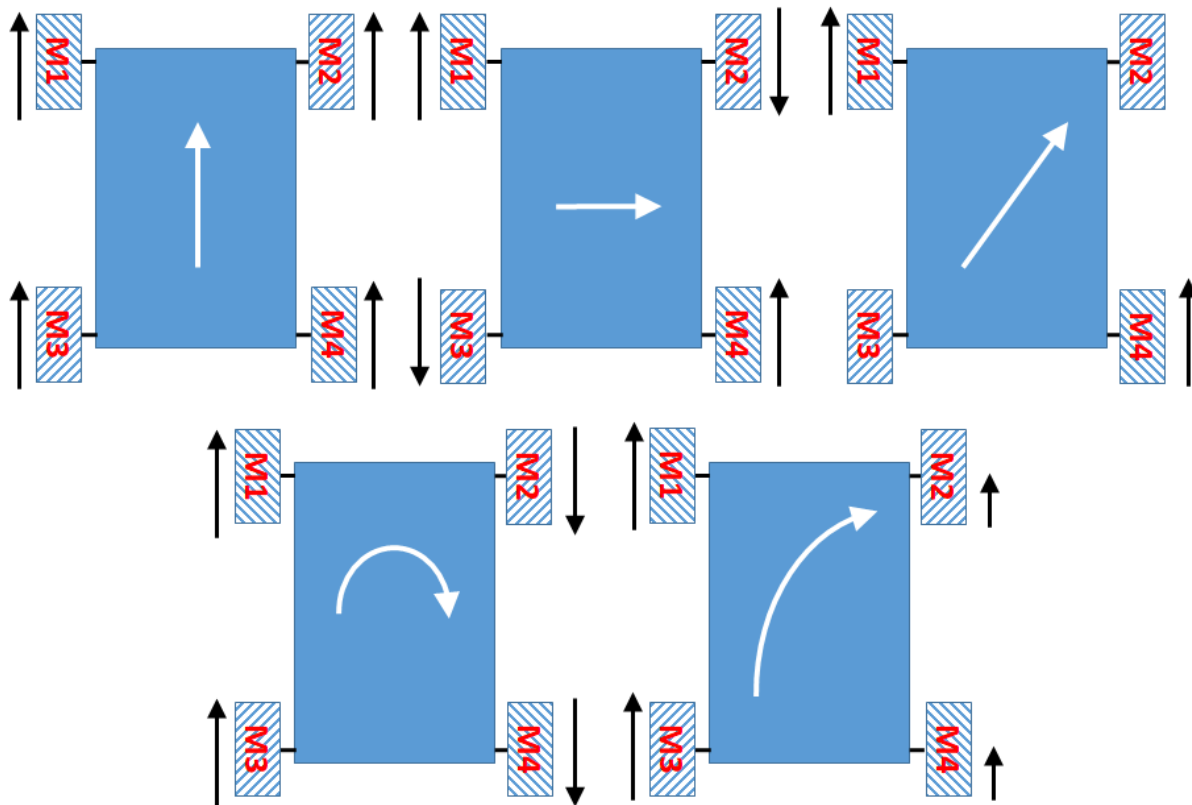
Build the Mecanum vehicle base model with sensors, according to the building instructions. The hubs of the Z20 toothed gears and the Mecanum wheels must be tightened well so there is no slippage while the vehicle is driving.

In this task, you will use the track sensor (front) and ultrasound distance sensor for obstacle detection.

The ultrasound distance sensor is connected to I6 (black cable), the two infrared (IR) sensors are connected to I7 (right sensor), and the yellow/blue cable to I8 (left sensor, blue cable). The IR sensors and the ultrasound sensor must also be supplied with power via the 9V voltage output of the TXT.

Use the interface test to check whether the four motors are connected correctly (forward: anti-clockwise), that the four encoders are connected to the correct counter inputs (M1: C1, M2: C2, M3: C3 and M4: C4) and the two IR sensors are delivering the right values (white surface: 1, black line: 0).

Programming tasks



Most important types of movement with Mecanum wheels

1. Synchronous drive in all directions

The figure shows the control of a Mecanum vehicle. The top three types of movements show the wheel drive required for travelling straight ahead, travelling at an angle and travelling sideways. Each of these three types of movement includes the forward and backward movement, as well as the movement to the left and right. If the black arrow beside the wheel is pointing upwards, the associated wheel should turn forwards.

Because of this, we can differentiate between the following eight movements of the Mecanum vehicle (see also the animation in [1]):

- Forwards
- Backwards
- Sideways to the left
- Sideways to the right
- At an angle to the front left
- At an angle to the front right

- At an angle to the rear left
- At an angle to the rear right

Develop a Blockly subroutine for each of these movements (a function) to which you can transmit the speed of the motors as a parameter.

The functions will be required in subsequent functions. These make the control programs clearer and easier to understand. Test your functions using simple example programs.

Please note: Synchronisation of the motors is particularly important in Mecanum wheels (see also task 6 of the Robotics TXT 4.0 Base Set).

2. Synchronous turning

The bottom two types of movement shown in the figure are the two most important rotational movements that can be completed with Mecanum wheels: turning in place and travelling along a curve. These also include multiple directions of movement, such as rotating to the left and right, or turning left or right and travelling forwards and backwards along a curve.

In total, this results in six additional movements:

- Rotating in place to the right (clockwise)
- Rotating in place to the left (anti-clockwise)
- Turning to the right
- Turning to the left
- Turning backwards to the right
- Turning backwards to the left

Develop a Blockly subroutine for each of these movements (a function) to which you can transmit the speed of the motors as a parameter. Test your functions using simple example programs.

3. Synchronous drive with distance specification

To navigate the Mecanum vehicle precisely, you will now need a section of the command set from programming tasks 1 and 2, each with a specified distance as well – the number of impulses by which the motor should turn.

3a. Add a specified distance to each of the following movement functions from programming tasks 1 and 2: a set number of impulses

- To travel straight ahead (forward/backward)
- To travel sideways (left/right) and
- To turn around the vehicle's own axis (left: clockwise, right: anti-clockwise).

3b. Test these six functions on a previously measured test track and use experiments to determine the factors (impulses per cm or impulses per °) to calculate the required impulses from

- A distance travelling straight ahead, in cm
- A distance travelling sideways, in cm and
- A rotational angle indicated in °

Similar to task 6 in the Robotics TXT 4.0 Base Set, you can measure the circumference of a Mecanum wheel as an initial estimate of the conversion factor when travelling straight ahead, and use it to derive the factor.

4. Line detection

The Mecanum vehicle should now identify the boundary lines and edges using the track sensor, and avoid these: If a (black) boundary line or gap is detected, the vehicle should drive back 10 cm, turn a quarter turn away from the edge (45°) and continue travelling.

4a. Draw a state diagram showing this.

4b. Now, use your navigation functions from programming tasks 1 and 2 to write a Blockly program (see also task 6 from the Robotics TXT 4.0 Base Set).

Experimental tasks

1. Obstacle detection using ultrasound

The Mecanum vehicle is equipped with an ultrasound sensor which delivers the distance from an object in cm (see also task 1 from the Robotics TXT 4.0 Base Set).

Write a Blockly program that prevents the vehicle from coming any closer than 15 cm to an obstacle. When it detects an obstacle, it should avoid it by driving sideways until the ultrasound sensor no longer detects an obstacle within a distance of 25 cm (see also task 6 in the Robotics TXT 4.0 Base Set), and then continue travelling.

2. Encoder navigation

You can also use the values from the encoder in the Mecanum vehicle to calculate the distance travelled. This will allow you to navigate to a specified goal, for which the linear distance in cm is specified. Mark a goal 3 m away (linear distance) on the floor for your test.

Now, block the path to the goal with first one, then later with multiple obstacles. Develop a strategy that will allow the vehicle to avoid the obstacles and then continue travelling to the specified goal along the shortest available path.

2a. Describe your strategy using a sketch.

2b. Add this function to your Blockly program from experimental task 1.

Tip: The Mecanum vehicle turns around the centre of the vehicle. All calculations to the path of travel, therefore, always relate to the centre point of the vehicle. If the travel should start and end on a line in front of the bumper, then the vehicle must finally be turned back in the direction of travel so that it will stop in front of the goal line.

Annex

Mecanum vehicle with obstacle detection

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection to transmit the program to the TXT4.0.
- Course sheet with black line in a closed circle, 2 cm wide (from the Robotics TXT 4.0 Base Set).

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](https://github.com/2605). github.io
- [2] Online diagram editor for creating state diagrams (Format drawio):
<https://www.diagrameditor.de/>

Task 1: Mecanum vehicle with obstacle detection

In this beginning task, students will learn how to navigate using a vehicle with Mecanum wheels. They will develop functions to abstract the movement processes. The vehicle will learn how to move in space and avoid obstacles and boundary lines.

Omniwheels allow a vehicle to move in any direction at any time. On a Mecanum wheel, the rollers are placed at an angle to the main axle, allowing for omnidirectional driving manoeuvres.

Topic

Controlling a vehicle with Mecanum wheels and detecting obstacles.

Learning objectives

- Understanding the function and control of Mecanum wheels
- Easy to understand programming with state variables and functions
- Detecting lines and obstacles using sensors (infrared, ultrasound)
- Navigation via motor pulses

Time required

students will need 45-90 minutes (one to two hours of instruction) to build the base model Mecanum vehicle with sensors using the building instructions, depending on their previous experience.

To develop the programs to solve the programming tasks, students will need experience with the Robotics TXT 4.0 Base Set, two to three hours of instruction (90-135 minutes).

Completing the experimental tasks will require an additional 90-135 minutes.

Annex

Task 1: Mecanum vehicle with obstacle detection

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection to transmit the program to the TXT4.0.
- Course sheet with black line in a closed circle, 2 cm wide (from the Robotics TXT 4.0 Base Set).

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](https://github.com/2605). github.io
- [2] Online diagram editor for creating state diagrams (Format drawio):
<https://www.diagrammeditor.de/>

Name: _____

Class: _____

Date: _____

Solution sheet task 1

Mecanum vehicle with obstacle detection

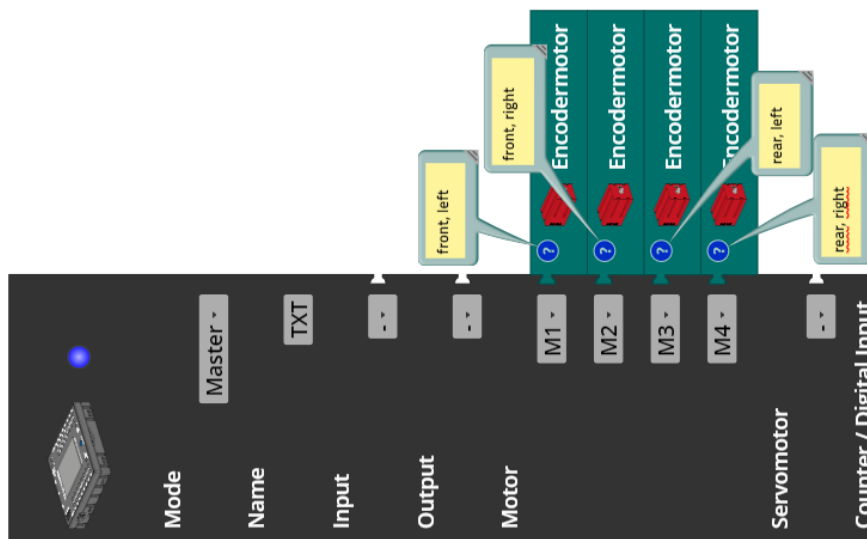
The control functions developed in the first programming task serve as the basis for all subsequent tasks.

The ways the sensors are used (track sensor, ultrasound) are similar to those in task 6 of the Robotics TXT 4.0 Base Set. Thanks to the flexible movement options offered by the Mecanum wheels, however, there are also simpler solutions available.

Programming tasks

1. Synchronous drive in all directions

Configuring the sensors:



Control functions (example):

```

+ define forward with:
  - variable: velocity
  + - set motor TXT_M1 speed ccw velocity
  sync with motor TXT_M2 direction ccw
  sync with motor TXT_M3 direction ccw
  sync with motor TXT_M4 direction ccw
    
```

```

+ define backward with:
  - variable: velocity
  + - set motor TXT_M1 speed cw velocity
  sync with motor TXT_M2 direction cw
  sync with motor TXT_M3 direction cw
  sync with motor TXT_M4 direction cw
    
```

```

+ define left with:
  - variable: velocity
  + - set motor TXT_M1 speed cw velocity
  sync with motor TXT_M2 direction ccw
  sync with motor TXT_M3 direction ccw
  sync with motor TXT_M4 direction cw
    
```

```

+ define right with:
  - variable: velocity
  + - set motor TXT_M1 speed ccw velocity
  sync with motor TXT_M2 direction cw
  sync with motor TXT_M3 direction cw
  sync with motor TXT_M4 direction ccw
    
```

```

+ define diag_left with:
  - variable: velocity
  + - stop motor TXT_M1
  sync with motor TXT_M4
  + - set motor TXT_M2 speed ccw velocity
  sync with motor TXT_M3 direction ccw
    
```

```

+ define diag_right with:
  - variable: velocity
  + - set motor TXT_M1 speed ccw velocity
  sync with motor TXT_M4 direction ccw
  + - stop motor TXT_M2
  sync with motor TXT_M3
    
```

```

+ define diag_backward_left with:
  - variable: velocity
  + - set motor TXT_M1 speed cw velocity
  sync with motor TXT_M4 direction cw
  + - stop motor TXT_M2
  sync with motor TXT_M3
    
```

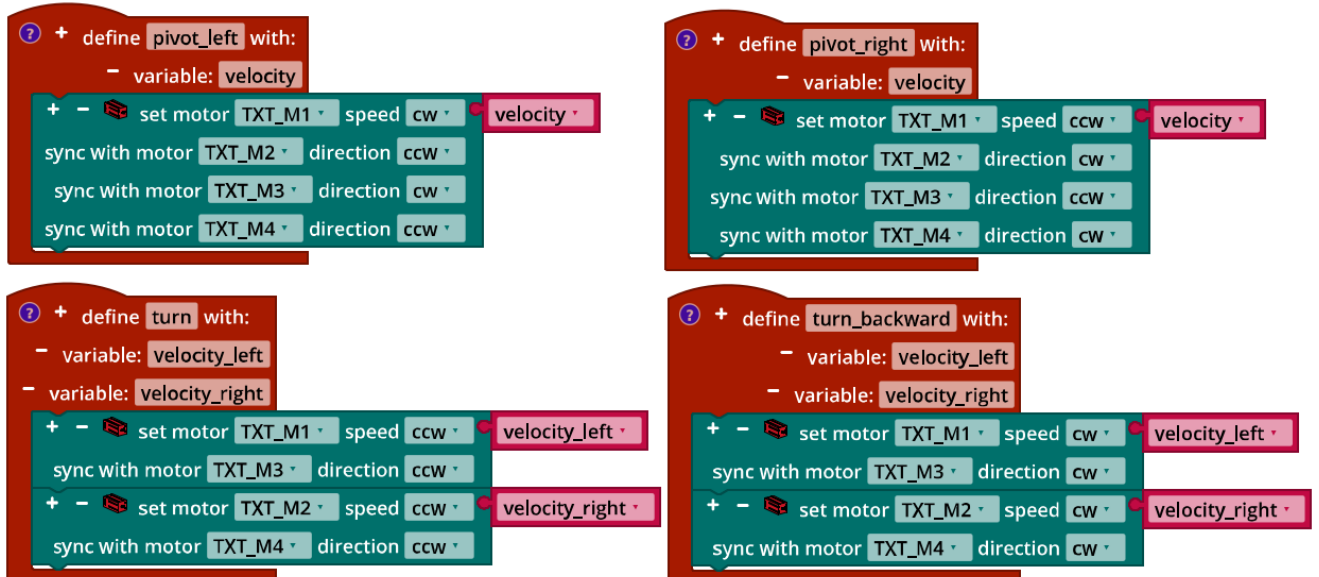
```

+ define diag_backward_right with:
  - variable: velocity
  + - stop motor TXT_M1
  sync with motor TXT_M4
  + - set motor TXT_M2 speed cw velocity
  sync with motor TXT_M3 direction cw
    
```

Mecanum_Synchronous_Driving_Funktionen.ft

2. Synchronous turning

Control functions (example):



```

+ define pivot_left with:
- variable: velocity
+ - set motor TXT_M1 speed cw velocity
sync with motor TXT_M2 direction ccw
sync with motor TXT_M3 direction cw
sync with motor TXT_M4 direction ccw

+ define pivot_right with:
- variable: velocity
+ - set motor TXT_M1 speed ccw velocity
sync with motor TXT_M2 direction cw
sync with motor TXT_M3 direction ccw
sync with motor TXT_M4 direction cw

+ define turn with:
- variable: velocity_left
- variable: velocity_right
+ - set motor TXT_M1 speed ccw velocity_left
sync with motor TXT_M3 direction ccw
+ - set motor TXT_M2 speed ccw velocity_right
sync with motor TXT_M4 direction ccw

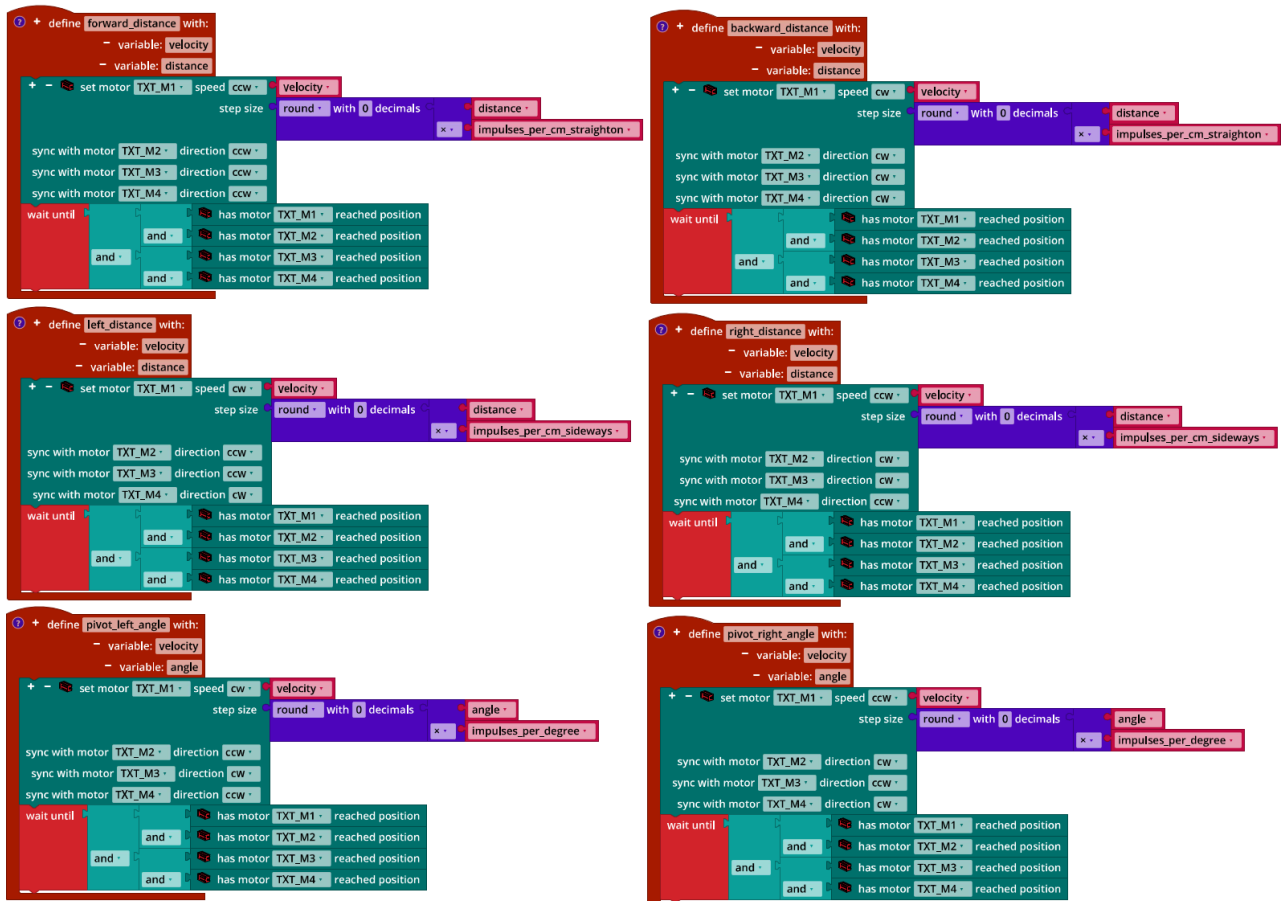
+ define turn_backward with:
- variable: velocity_left
- variable: velocity_right
+ - set motor TXT_M1 speed cw velocity_left
sync with motor TXT_M3 direction cw
+ - set motor TXT_M2 speed cw velocity_right
sync with motor TXT_M4 direction cw
    
```

Mecanum_Synchronous_Turning_Functions.ft

3. Synchronous drive with distance specification

3a. The distance (in cm) can be converted into impulses either in the function or when it is called up.

Program functions (example):



```

define forward_distance with:
  - variable: velocity
  - variable: distance
  + set motor TXT_M1 speed ccw velocity
  step size round with 0 decimals distance
  impulses_per_cm_straighton
  sync with motor TXT_M2 direction ccw
  sync with motor TXT_M3 direction ccw
  sync with motor TXT_M4 direction ccw
  wait until
    and has motor TXT_M1 reached position
    and has motor TXT_M2 reached position
    and has motor TXT_M3 reached position
    and has motor TXT_M4 reached position

define backward_distance with:
  - variable: velocity
  - variable: distance
  + set motor TXT_M1 speed cw velocity
  step size round with 0 decimals distance
  impulses_per_cm_straighton
  sync with motor TXT_M2 direction cw
  sync with motor TXT_M3 direction cw
  sync with motor TXT_M4 direction cw
  wait until
    and has motor TXT_M1 reached position
    and has motor TXT_M2 reached position
    and has motor TXT_M3 reached position
    and has motor TXT_M4 reached position

define left_distance with:
  - variable: velocity
  - variable: distance
  + set motor TXT_M1 speed cw velocity
  step size round with 0 decimals distance
  impulses_per_cm_sideways
  sync with motor TXT_M2 direction ccw
  sync with motor TXT_M3 direction ccw
  sync with motor TXT_M4 direction cw
  wait until
    and has motor TXT_M1 reached position
    and has motor TXT_M2 reached position
    and has motor TXT_M3 reached position
    and has motor TXT_M4 reached position

define right_distance with:
  - variable: velocity
  - variable: distance
  + set motor TXT_M1 speed ccw velocity
  step size round with 0 decimals distance
  impulses_per_cm_sideways
  sync with motor TXT_M2 direction cw
  sync with motor TXT_M3 direction cw
  sync with motor TXT_M4 direction ccw
  wait until
    and has motor TXT_M1 reached position
    and has motor TXT_M2 reached position
    and has motor TXT_M3 reached position
    and has motor TXT_M4 reached position

define pivot_left_angle with:
  - variable: velocity
  - variable: angle
  + set motor TXT_M1 speed cw velocity
  step size round with 0 decimals angle
  impulses_per_degree
  sync with motor TXT_M2 direction ccw
  sync with motor TXT_M3 direction cw
  sync with motor TXT_M4 direction ccw
  wait until
    and has motor TXT_M1 reached position
    and has motor TXT_M2 reached position
    and has motor TXT_M3 reached position
    and has motor TXT_M4 reached position

define pivot_right_angle with:
  - variable: velocity
  - variable: angle
  + set motor TXT_M1 speed ccw velocity
  step size round with 0 decimals angle
  impulses_per_degree
  sync with motor TXT_M2 direction cw
  sync with motor TXT_M3 direction ccw
  sync with motor TXT_M4 direction cw
  wait until
    and has motor TXT_M1 reached position
    and has motor TXT_M2 reached position
    and has motor TXT_M3 reached position
    and has motor TXT_M4 reached position
  
```

Mecanum_Synchronous_Navigation_Distance.ft

3b. The circumference of a Mecanum wheel is around 19 cm. This makes it easy to calculate the number of impulses per revolution (as in task 6 of the Robotics TXT 4.0 Base Set): Since the wheels are geared down by the motors with a ratio of 1:2, this is around $\frac{63.9 \cdot 2}{19} \approx 6.726$ impulses/cm.

Tests over distances of several meters show that the value must be corrected to around **6.82 impulses/cm**.

Only experiments (see the following program) can help determine the conversion factor for sideways travel and turning. In tests, the value for sideways travel was calculated

at around **9.5 impulses/cm** and the value for turning was calculated at **1.7 impulses/angular degree**.

Program (example) for testing the conversion factors:

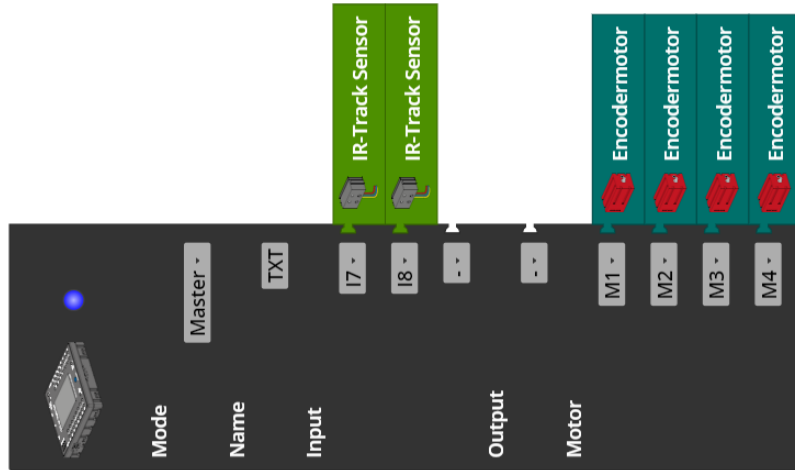
```

program start
  set speed to 512
  set dist to 50
  set turn to 180
  set impulses_per_cm_straighton to 6.82
  set impulses_per_cm_sideways to 9.5
  set impulses_per_degree to 1.7
  repeat forever
    do forward_distance with:
      velocity speed
      distance dist
      wait s 1
    do backward_distance with:
      velocity speed
      distance dist
      wait s 1
    do left_distance with:
      velocity speed
      distance dist
      wait s 1
    do right_distance with:
      velocity speed
      distance dist
      wait s 1
    do pivot_left_angle with:
      velocity speed
      angle turn
      wait s 1
    do pivot_right_angle with:
      velocity speed
      angle turn
      wait s 1
  
```

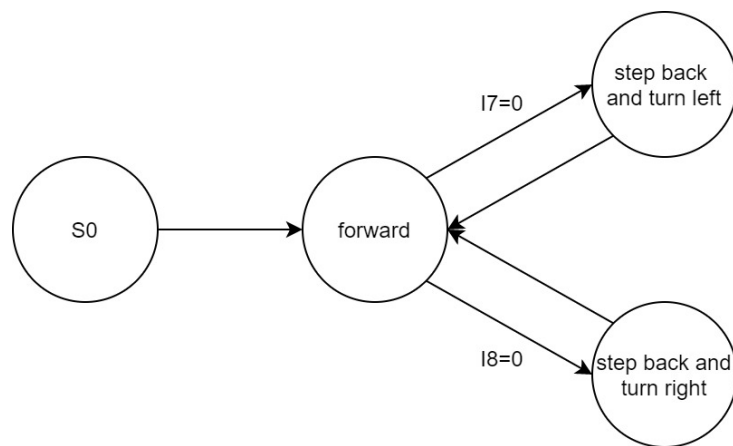
Mecanum_Synchronous_Navigation_Distance.ft

4. Line detection

Configuring the sensors and actuators:



4a. State diagram:



State-Transition_Diagram_Mecanum_with_IR_Sensors.drawio

4b. Program excerpt (example):

```

program start
  set speed to 512
  set dist to 10
  set turn to 45
  set impulses_per_cm_straighton to 6.82
  set impulses_per_degree to 1.7
  forward with:
    velocity speed
  repeat forever
  do + if is IR track sensor TXT_I8 state = 0
  do stop
  backward_distance with:
    velocity speed
    distance dist
  pivot_right_angle with:
    velocity speed
    angle turn
  forward with:
    velocity speed
  else if - is IR track sensor TXT_I7 state = 0
  do stop
  backward_distance with:
    velocity speed
    distance dist
  pivot_left_angle with:
    velocity speed
    angle turn
  forward with:
    velocity speed
  
```

```

+ define forward with:
  - variable: velocity
  + - set motor TXT_M1 speed ccw velocity
  sync with motor TXT_M2 direction ccw
  sync with motor TXT_M3 direction ccw
  sync with motor TXT_M4 direction ccw
  
```

```

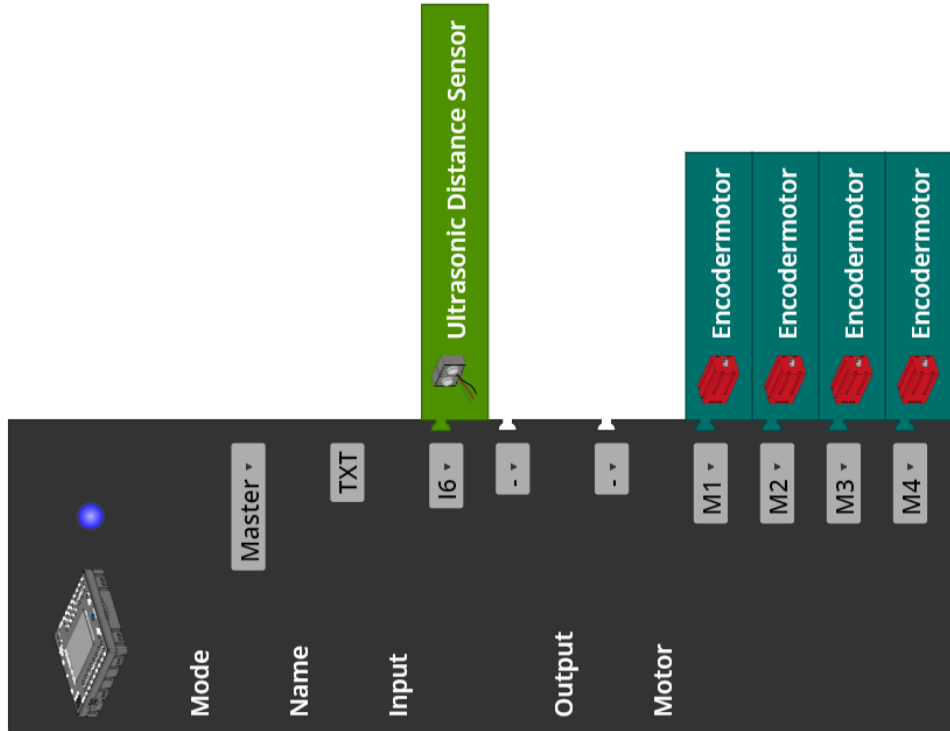
+ define stop
  + - stop motor TXT_M1
  sync with motor TXT_M2
  sync with motor TXT_M3
  sync with motor TXT_M4
  
```

Mecanum_Boundary_Line.ft

Experimental tasks

1. Obstacle detection using ultrasound

Configuring the sensors and actuators:



Program excerpt (example):

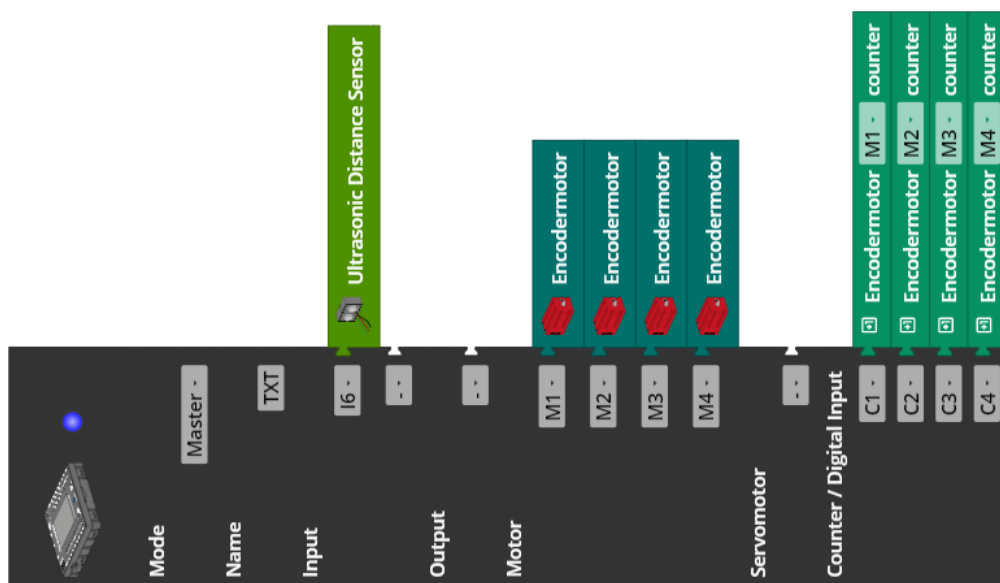
```

program start
  set speed to 512
  set obstacle_distance to 15
  set no_obstacle to 25
  set deviation to 10
  set impulses_per_cm_sideways to 9.5
  wait ms 250
  forward with:
    velocity speed
  repeat forever
  do + if is ultrasonic sensor TXT_I6 distance ≤ obstacle_distance
  do
    right with:
      velocity speed
    wait until is ultrasonic sensor TXT_I6 distance ≥ no_obstacle
    right_distance with:
      velocity speed
      distance deviation
    forward with:
      velocity speed
  
```

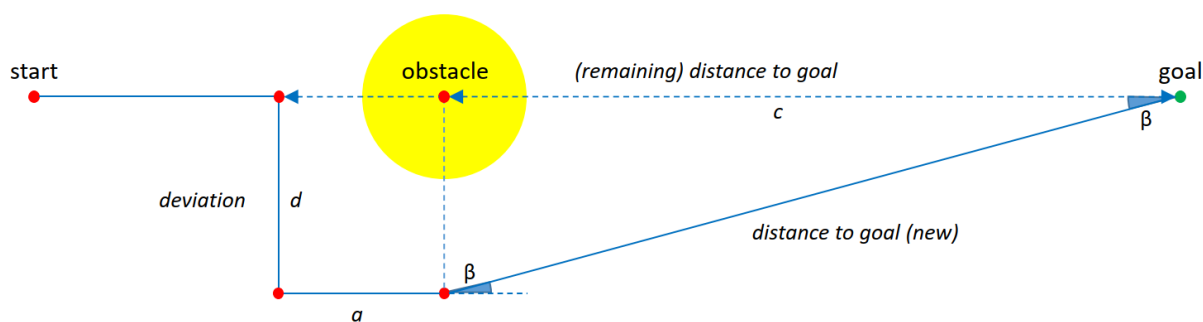
Mecanum_Collision_Prevention.ft

2. Encoder navigation

Configuring the sensors and actuators:



2a. There are multiple strategies for solving this task. The strategy illustrated in the following uses a simplified version of the mathematical model described in the example solution for the experimental task in task 6 of the Robotics TXT 4.0 Base Set, thanks to the possibilities offered by the Mecanum wheels:



Mecanum_Model_Target_Reacher.jpg

Solution strategy:

The Mecanum vehicle is aligned towards the *goal* and started travelling straight ahead. If it detects an *obstacle* while driving, it avoids it by moving to the right side until the obstacle no longer detects any obstacles within a distance of 25 cm. To ensure the robot does not hit the obstacle when driving past, it drives another 15 cm to the right (*d*).

Then it drives 25 cm (*a*) straight ahead. Then, based on the remaining impulses for the (original) direct path to the goal (*c*) and the distance travelled to drive around the obstacle sideways (*d*) the new distance to the goal (*distance to goal (new)*) and the angle of rotation β for correcting alignment to the goal are calculated.

Calculating the new distance to the goal is simple; it can be obtained using the Pythagorean theorem:

$$distance\ to\ goal\ (new) = \sqrt{c^2 + d^2}$$

Therefore, we calculate the distance *d* from converting the impulses counted during sideways travel plus 15 cm; the length *c* is calculated by subtracting the distance to drive around the obstacle *a* (25 cm) from the impulses still to be completed at the start of the deviating manoeuvre (*remaining impulses to goal*) and the distance this represents (*remaining distance to goal*).

The angle β by which the robot must turn to the left so it can drive towards the goal once again can also be determined in a calculation step:

$$\beta = \arccos\left(\frac{c}{distance\ to\ goal\ (new)}\right)$$

The solution strategy also works with multiple obstacles in a row.

After reaching the goal, the vehicle turns back to the right to the original direction of travel. To do so, all of the angles β by which the Mecanum vehicle had to change its direction after the avoidance manoeuvres are added up.

Important note: Since the conversion values for the impulses per cm differ during forward travel and sideways travel, the calculation of the new distance to the goal and angle of rotation β should be completed in cm.

2b. Program excerpt (example):

```

program start
set speed to 512
set impulses_per_cm_straighton to 6.82
set impulses_per_cm_sideways to 9.5
set impulses_per_degree to 1.7
set distance_to_goal to 300
set impulses_to_goal to round with 0 decimals distance_to_goal x impulses_per_cm_straighton
set obstacle_distance to 15
set no_obstacle to 25
set deviation to 15
set a to 25
set turns to 0
forward with:
velocity speed
repeat while is counter TXT_C1 value < impulses_to_goal
do + if is ultrasonic sensor TXT_I6 distance <= obstacle_distance
do stop
wait ms 250
change impulses_to_goal by - get counter TXT_C1 value
right with:
velocity speed
wait until is ultrasonic sensor TXT_I6 distance >= no_obstacle
stop
wait ms 250
set d to get counter TXT_C1 value
+ impulses_per_cm_sideways
+ deviation
right_distance with:
velocity speed
distance deviation
set c to impulses_to_goal
÷ impulses_per_cm_straighton
- a
forward_distance with:
velocity speed
distance a
set distance_to_goal to square root square c
+ square d
set beta to acos c ÷ distance_to_goal
change turns by beta
pivot_left_angle with:
velocity speed
angle beta
set impulses_to_goal to round with 0 decimals distance_to_goal
x impulses_per_cm_straighton
forward with:
velocity speed
pivot_right_angle with:
velocity speed
angle turns
    
```

Mecanum_Target_Reacher.ft

Annex

Mecanum vehicle with obstacle detection

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection to transmit the program to the TXT4.0.
- Course sheet with black line in a closed circle, 2 cm wide (from the Robotics TXT 4.0 Base Set).

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](https://github.com/2605-robotics/How-a-Mecanum-Drive-Works). github.io
- [2] Online diagram editor for creating state diagrams (Format drawio):
<https://www.diagrammeditor.de/>

Name: _____

Class: _____

Date: _____

Task 2

Track follower

Construction task

For the third programming task and the experimental tasks, we will need the camera on the Mecanum vehicle base model, with sensors. Connect it to the USB1 port of the TXT.

Important note: When the TXT is started, the servo automatically switches to “forward setting”. Then, insert the servo lever so that the camera is tilted at an angle of approximately 45° from vertical to the front.

Note: If you move the servo manually with the TXT switched on, you may damage it.

Programming tasks

1. Track follower with track sensor

Now, the Mecanum vehicle should follow the black, approx. 2 cm wide track on the course from the Robotics TXT 4.0 Base Set (see also task 8 from the Robotics TXT 4.0 Base Set) with the help of the track sensor.

1a. First, create a state diagram that describes the possible states and behaviour of the Mecanum vehicle.

1b. Convert your state diagram into a Blockly program. To do so, use a state variable with a state value which you determine based on the values from the IR sensors.

Test the program on the circular course of the Robotics TXT 4.0 Base Set.

1c. How can you increase the speed of the Mecanum track follower? Conduct tests and measure the time the track follower needs to complete the course.

Solution variants (adjustments, parameters)	Time



2. Track follower with obstacle detection

There can also be obstacles on the track. If the ultrasound sensor detects an obstacle at a maximum 2 cm distance on front of the vehicle, the vehicle should move sideways to travel past the obstacle and then find the track once again.

Think of different possible solutions to find the line. Add this function to your Blockly program and test the variants. What advantages and disadvantages do they have?

3. Track follower with colour control

You will see four coloured surfaces at the corners of the course. You can use these coloured surfaces to give commands to your Mecanum vehicle.

3a. Add a colour recognition function to your Blockly program. Output the RGB-HEX colour code of the detected colour on the console in order to calibrate the detection.

3b. Determine how the Mecanum vehicle should react to different colours (at least two), and modify your Blockly program accordingly.

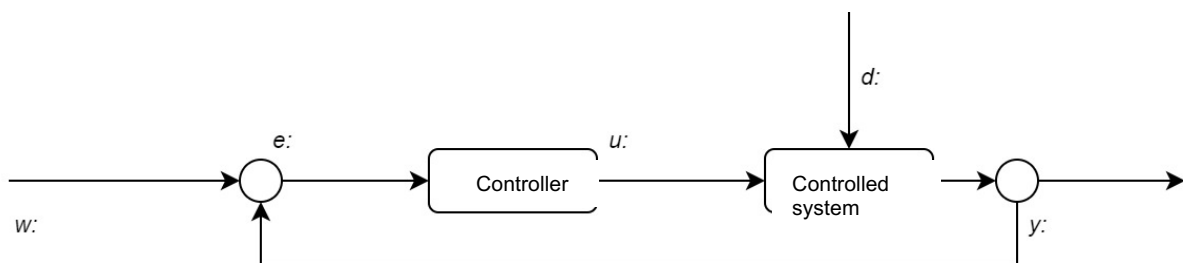
Note: Do not be surprised if the HEX value displayed on the console seems to fluctuate significantly – the hue value is stable. You can use any of the displayed HEX colour codes for the detection.

Experimental tasks

1. Track follower with proportional controller

Now, you should equip the track follower with a proportional controller (P controller) like the track follower buggy from task 8 of the Robotics TXT 4.0 Base Set. The amount by which the direction is corrected should depend on the amount of line deviation. Here as well, the camera's line detection function should be used to determine deviation from the track.

1a. First, label the following control circuit with the values for your Mecanum track follower:



1b. Design and program a proportional controller (P controller) for your Mecanum track follower. You can use the detection process for the black line from experimental task 1 from task 8 of the Robotics TXT 4.0 Base Set.

Test your program on the simple straight line course by starting the Mecanum track follower in parallel to the track, so that the centre point of the track appears far to the left of the image.

Tip: Start with the proportionality factor $k_p = 2$. Increase the value in increments of 1 until the Mecanum track follower quickly “engages”, meaning that the deviation from the track quickly reduces without the controller jumping or overshooting too severely.

1c. Add a text output to your program that indicates

- the time (in ms) which has passed since the start of the program and
- the value for the current deviation

separated by a space on the console after each change in the deviation from the track.

After each test run with a different value for k_p , copy the information output on the console into a spreadsheet, and display the values graphically in a diagram (x: Time, y: Deviation from the track). Adjust the proportionality factor k_p until the curve progression engages quickly.

Test the track follower with the circular course from the Robotics TXT 4.0 Base Set. You may need to reduce the maximum speed somewhat.

2. Track follower with PD controller

Just like the buggy in task 8 of the Robotics TXT 4.0 Base Set, you can add a “D” factor (differential factor) to the controller that takes the amount of change in deviation from the track into account in the speed correction, to further dampen overshooting.

Expand the P controller in your Mecanum track follower from experimental task 1 so that it is a PD controller. Complete test runs with different values for the differential factor k_d and use a spreadsheet program to display the data graphically.

Tip: Start your tests with the differential factor $k_d = 0.25$ and increase its value in increments of 0.25 until the overshoot of the P controller is dampened.

Annex

Task 2: Track follower

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection to transmit the program to the TXT4.0.
- Course sheet with straight black line 2 cm wide
- Course sheet with black line in a closed circle, 2 cm wide (from the Robotics TXT 4.0 Base Set)
- Obstacle (box, can, ...)

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](https://github.com/2605). github.io
- [2] Wikipedia: [Finite automaton \(state machine\)](#)
- [3] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme: [Modeling Software with Finite State Machines. A Practical Approach](#). Auerbach Publications, 2006.
- [4] Online diagram editor for creating state diagrams (Format drawio): <https://www.diagrammeditor.de/>
- [5] Wikipedia: [Control technology](#).
- [6] Wikipedia: [Controller](#).
- [7] RN-Wissen: [Control technology](#).
- [8] Tim Wescott: [PID without a PhD](#). Embedded Systems Programming, 10/2000, p. 86-108.

Task 2: Track follower

In this task, you will build a “track follower” using the track sensor from the Mecanum vehicle: The vehicle learns to travel autonomously along a black line.

In the experimental task, the vehicle will be outfitted with a camera with line detection: As an analogue sensor, it allows for a track follower with P and PD controller.

The task builds on task 8 from the Robotics TXT 4.0 Base Set.

Topic

Digital control of the vehicle and proportional (and PD) control of travel along a black line; identifying and reacting to obstacles.

Learning objectives

- Simple three point control using digital sensors
- Adding obstacle detection (ultrasound distance measurement)
- Analogue regulation using line detection (camera with image evaluation)
- Calibration of a P and PD controller

Time required

The Mecanum base model vehicle with sensors constructed in task 1 is used in this task.

Students will need the knowledge gained in the Robotics TXT 4.0 Base Set to develop the programs needed to solve the programming tasks (in particular task 8); assumed time 45 - 90 minutes (one to two hours of instruction).

A P and a PD controller are developed during the experimental tasks. Several hours of instruction should be set aside to program and set the controllers (90 - 180 minutes each). We recommend that students work in groups.

Annex

Task 2: Track follower

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection to transmit the program to the TXT4.0.
- Course sheet with straight black line 2 cm wide
- Course sheet with black line in a closed circle, 2 cm wide (from the Robotics TXT 4.0 Base Set)
- Obstacle (box, can, ...)

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](https://github.com/2605). github.io
- [2] Wikipedia: [Finite automaton \(state machine\)](#)
- [3] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme: [Modeling Software with Finite State Machines. A Practical Approach](#). Auerbach Publications, 2006.
- [4] Online diagram editor for creating state diagrams (Format drawio): <https://www.diagrammeditor.de/>
- [5] Wikipedia: [Control technology](#).
- [6] Wikipedia: [Controller](#).
- [7] RN-Wissen: [Control technology](#).
- [8] Tim Wescott: [PID without a PhD](#). Embedded Systems Programming, 10/2000, p. 86-108.

Name: _____

Class: _____

Date: _____

Solution sheet task 2

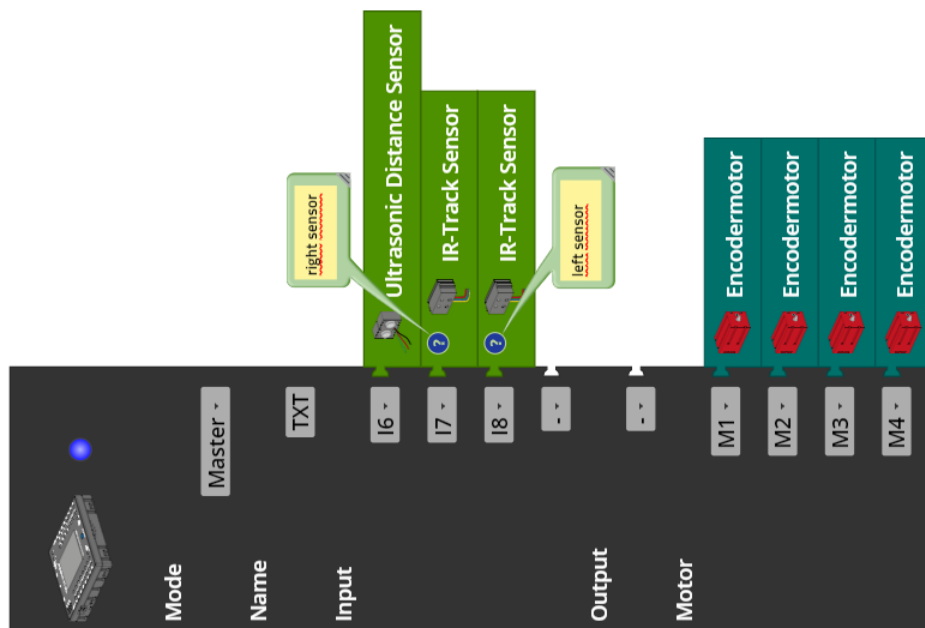
Track follower

The students' solutions should use the sub-functions from task 1 to control the vehicle. Just as in the solution for task 8 in the Robotics TXT 4.0 Base Set – a state variable should be used to differentiate between the states when programming the track follower. This makes the programs very clear and easy to understand.

The two experimental tasks help students understand how to develop and configure a P and PD controller. The output and graphic display of measured values are particularly important.

Construction task

Connecting the sensors:

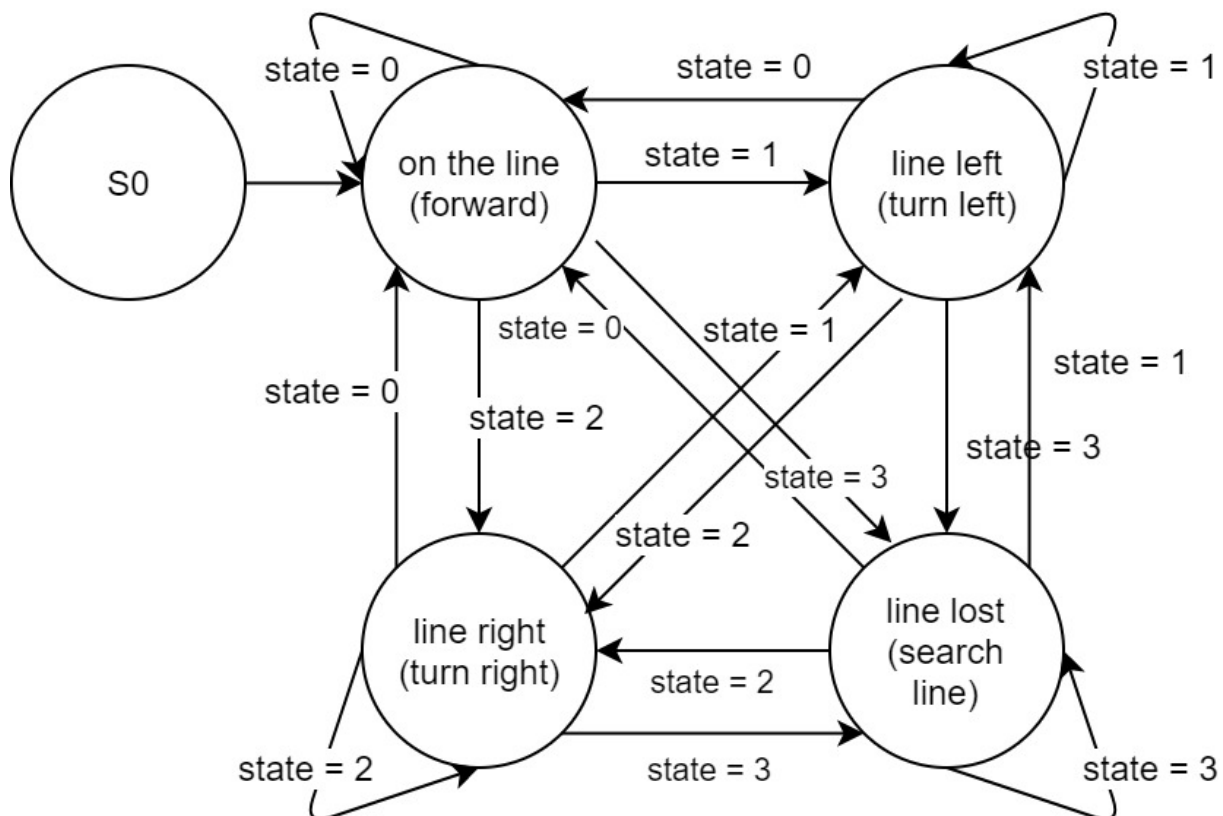


Programming tasks

1. Track follower with track sensor

Just like the buggy in task 8 of the Robotics TXT 4.0 Base Set, the Mecanum vehicle should be steered so that both IR sensors in the track sensor are centred over the black line, so that they both deliver a value of 0.

1a. State diagram for the (digital) track follower:



State-Transition_Diagram_Line_Follower.drawio

1b. Depending on the values from the left and right IR sensors, the state variables *state* have the following values:

- *state* = 0 → on the track (both sensors deliver a value of 0)
- *state* = 1 → track to the left (only the left sensor has the value 0)
- *state* = 2 → track to the right (only the right sensor has the value 0)
- *state* = 3 → track lost (both sensors deliver the value 1)

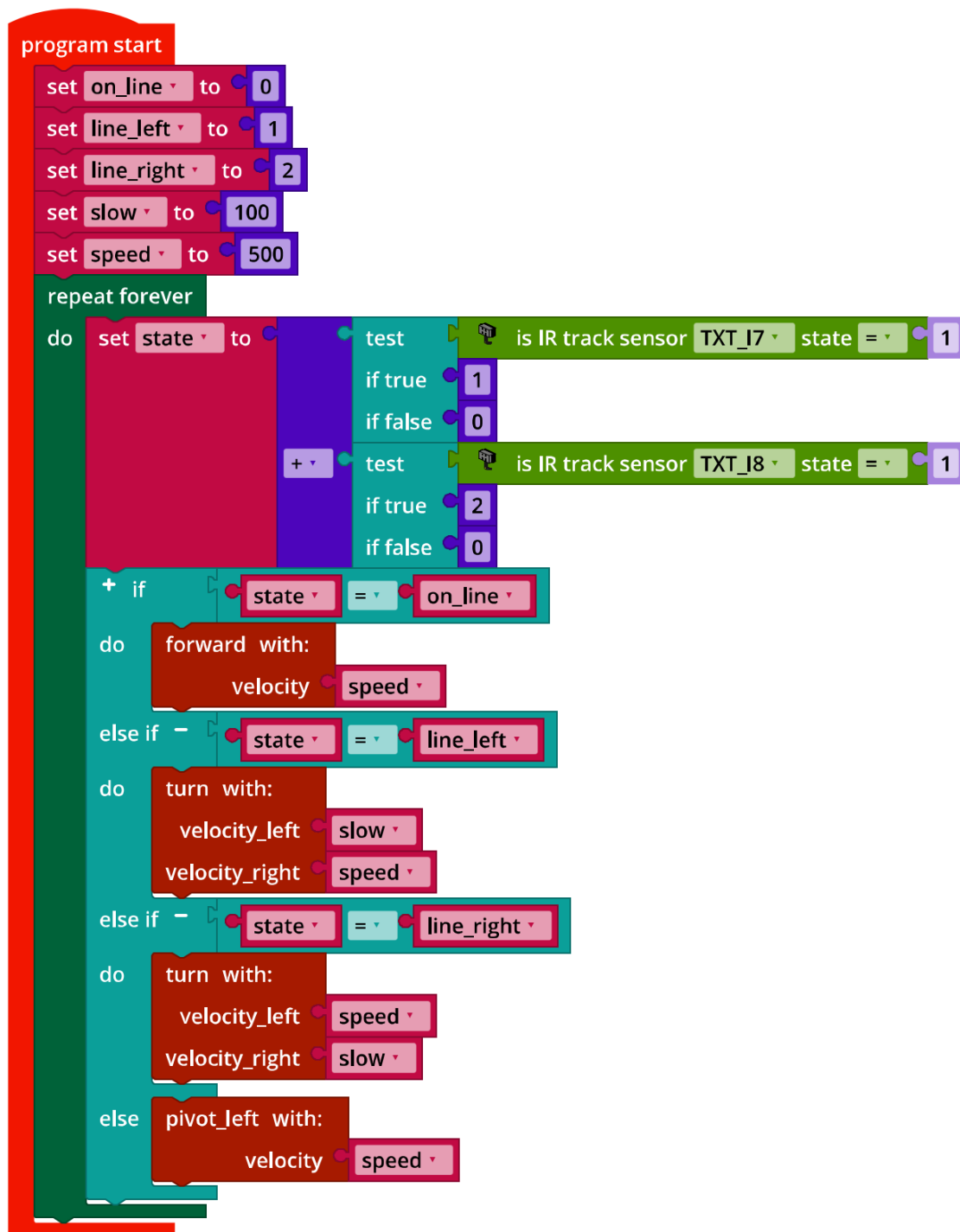
	I8 (left sensor)	I7 (right sensor)
Black line (track)	0 → <i>state</i> += 0	0 → <i>state</i> += 0
Light area	1 → <i>state</i> += 2	1 → <i>state</i> += 1

Reading example: If the left sensor (I8) delivers the value 0 and the right sensor (I7) delivers the value 1, then the track is located to the left of the centre of the vehicle. The state variable *state* has a value of 1, and in this state the vehicle must be steered to the left so that the sensor is centred over the track once again.

The following example solution uses the navigation functions programmed in task 1 for the Mecanum vehicle. This makes the program very easy to understand.

At the start of the loop, the state variable is set to a state value from {0, 1, 2, 3} depending on the values of the two IR sensors.

Program excerpt (example):



Mecanum_Line_Follower_digital.ft

1c. The speed of the track follower can be increased by choosing the highest possible main speed (“speed”) and reducing the speed distance between the motors during

steering, adjusting the “slow” speed so that the track follower does not lose the track, particularly on curves.

Note: You can also understand the digital track follower as a three-point controller which regulates the speed of the motors depending on the three states “left of the track”, “on the track” and “right of the track”. The area within which both IR sensors deliver the target value of “0” (both are directly over the track) is also called the *hysteresis*.

2. Track follower with obstacle detection

After the vehicle deviates from the line, it can find it again behind the obstacle by

- driving to the side once again (disadvantage: the depth of the obstacle is unknown, therefore it may hit the obstacle or miss the line if there is a tight curve right behind the obstacle; advantage: the vehicle finds a straight line in the right position)
- travelling diagonally (disadvantage: it may hit the obstacle; it is very likely to miss the line; advantage: a straight line is found in the correct alignment to continue driving)
- turning by 45°-90°, then driving straight ahead (disadvantage: it may hit the obstacle, miss the line or continue driving in the wrong direction once it finds the line)
- Driving in a curve around the obstacle (disadvantage: the vehicle may continue driving in the wrong direction once it finds the line)

Program excerpt (example):

```

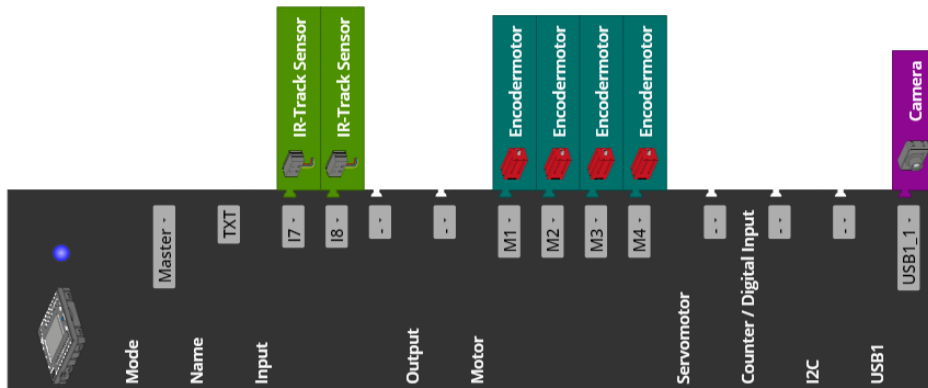
program start
set on_line to 0
set line_left to 1
set line_right to 2
set slow to 100
set speed to 500
set obstacle_distance to 6
set impulses_per_cm_straighton to 6.82
set impulses_per_cm_sideways to 9.5
set deviation to 15
set passing to 35
...
+ if is ultrasonic sensor TXT_I6 distance ≤ obstacle_distance
do
  right_distance with:
    velocity speed
    distance deviation
  forward_distance with:
    velocity speed
    distance passing
  left with:
    velocity speed
  wait until is IR track sensor TXT_I8 state = 0

```

Mecanum_Line_Follower_Obstacle_digital.ft

3. Track follower with colour control

Connecting the camera:



3b. Program excerpt (example):

```

on color color_detector detected: event
  set color_detected to 1
  + if is color event = hue tolerance 40 degree
  do set color to red
  else if - is color event = hue tolerance 40 degree
  do set color to green
  else if - is color event = hue tolerance 40 degree
  do set color to blue
  else set color to 0

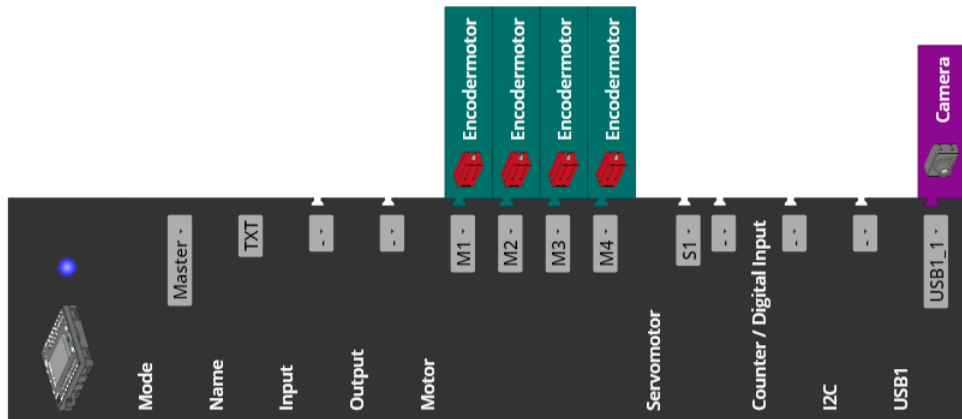
  set color_detected to 0
  set red to 1
  set green to 2
  set blue to 3

  + if color_detected = 1
  do + if color = red
  do 04_Braking.wav start playing audio file
  else if - color = green
  do 20_Motor_starting.wav start playing audio file
  else if - color = blue
  do 06_Car_horn_short.wav start playing audio file
  set color_detected to 0
  
```

Mecanum_Line_Follower_with_Color_Detection_digital.ft

Experimental tasks

Connecting the sensors:



1. Track follower with proportional controller

The camera line detector delivers the deviation from the centre of the detection window.

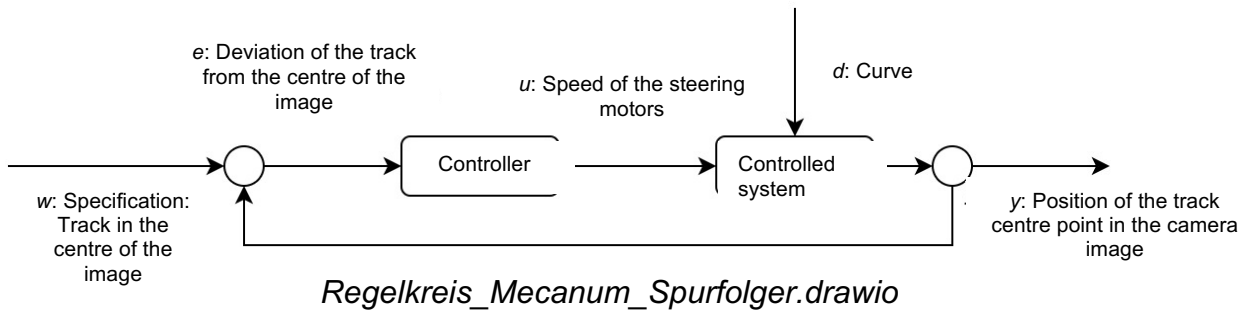
Configuring the line detection:

Name	Line	Position	Width	Red	Green	Blue
line_detector	1	-18	41	13	13	13

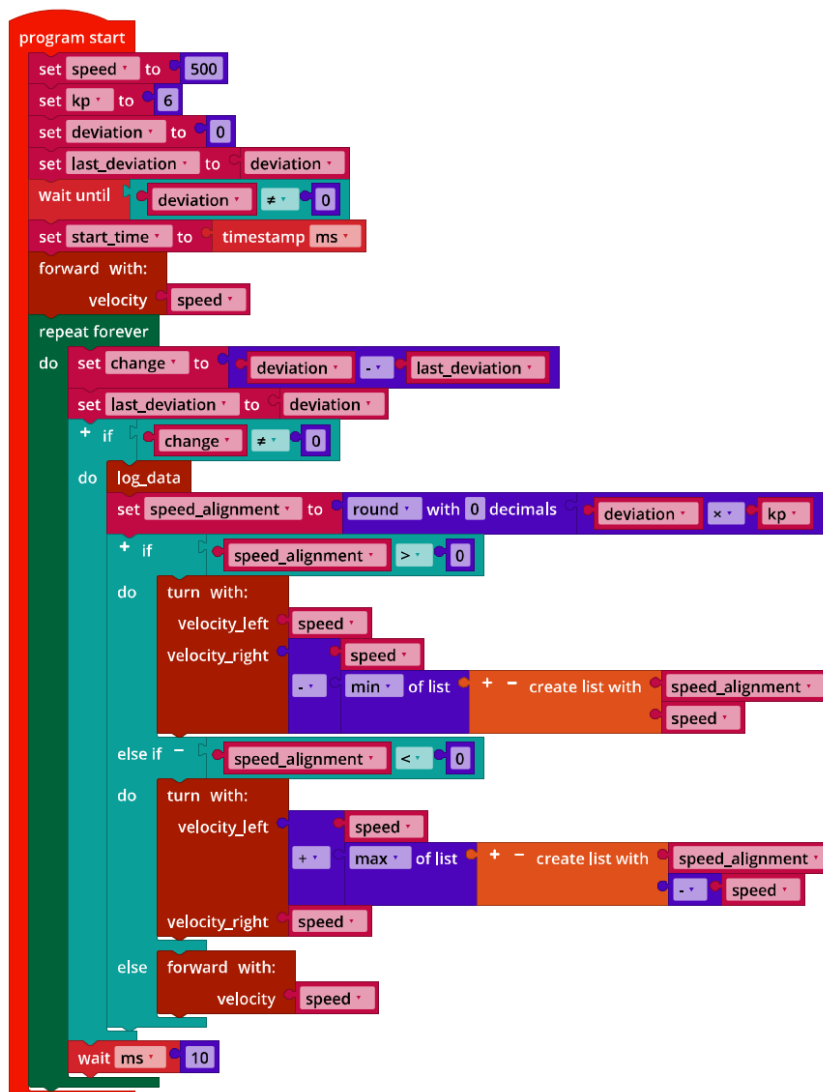
ROBOTICS Add On: Omniwheels – Secondary level I+II

Since coloured surfaces can be detected as a line, the line detection should be set to at least two lines.

1a. Control circuit:



1b. Program excerpt (example):



Mecanum_Line_Follower_P_Controller.ft

```

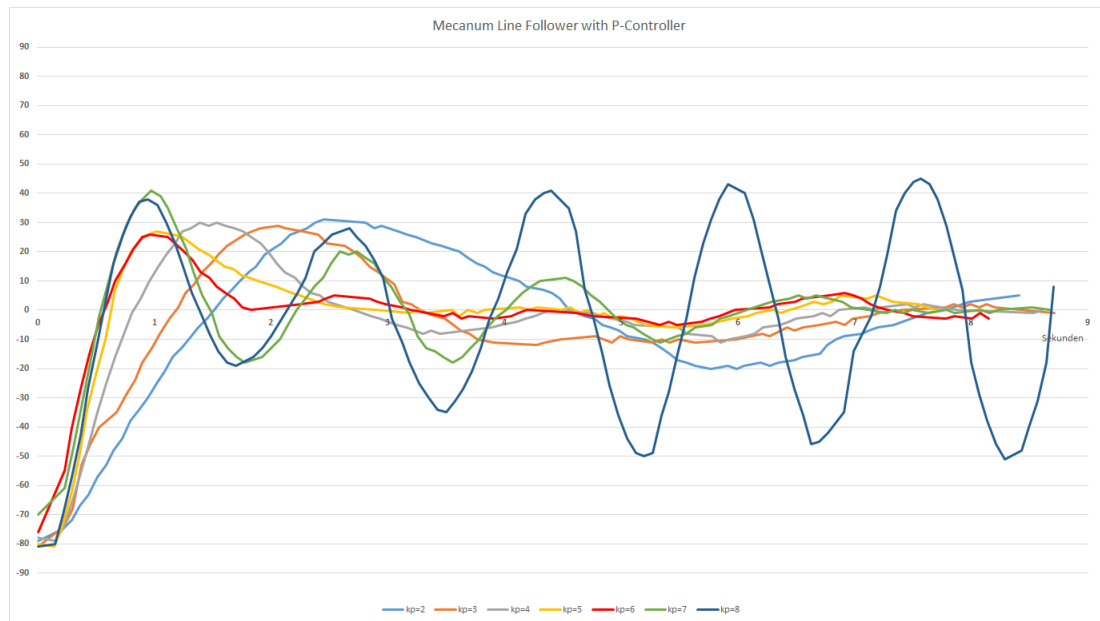
on lines line_detector detected: event list
  set index to 1
  repeat while index ≤ length of event
  do
    set line_color to get color of line index from event list as RGB
    + if
      in list line_color get # 1 < 100
      and
      in list line_color get # 2 < 100
      and
      in list line_color get # 3 < 100
    do
      set deviation to get position of line index from event list
      break out of loop
    change index by 1

+ define log_data
  print + create text with round with 0 decimals timestamp ms
  " "
  deviation
  
```

Mecanum_Line_Follower_P_Controller.ft

Multiplied by the proportionality factor k_p , the position is added to or subtracted from the motor speed for turning left or right (function *turn*), so that the deviation from the track is reduced with the change in the direction of travel. Depending on the vertical position of the line detector in the camera configuration, it may be necessary to adjust the minimum and maximum line width in the program.

1c. Measurement results for the P controller (with $k_p \in \{2, 3, 4, 5, 6, 7, 8\}$):



Mecanum_Line_Follower_with_P_Controller_Results.jpg

The red line with $k_p = 6$ engages most quickly after approx. 1.7 seconds, without overshooting a second time. The controller oscillates with $k_p = 8$ (dark blue line).

2. Track follower with PD controller

The D factor of the PD controller is the change in deviation multiplied by the differential factor k_d .

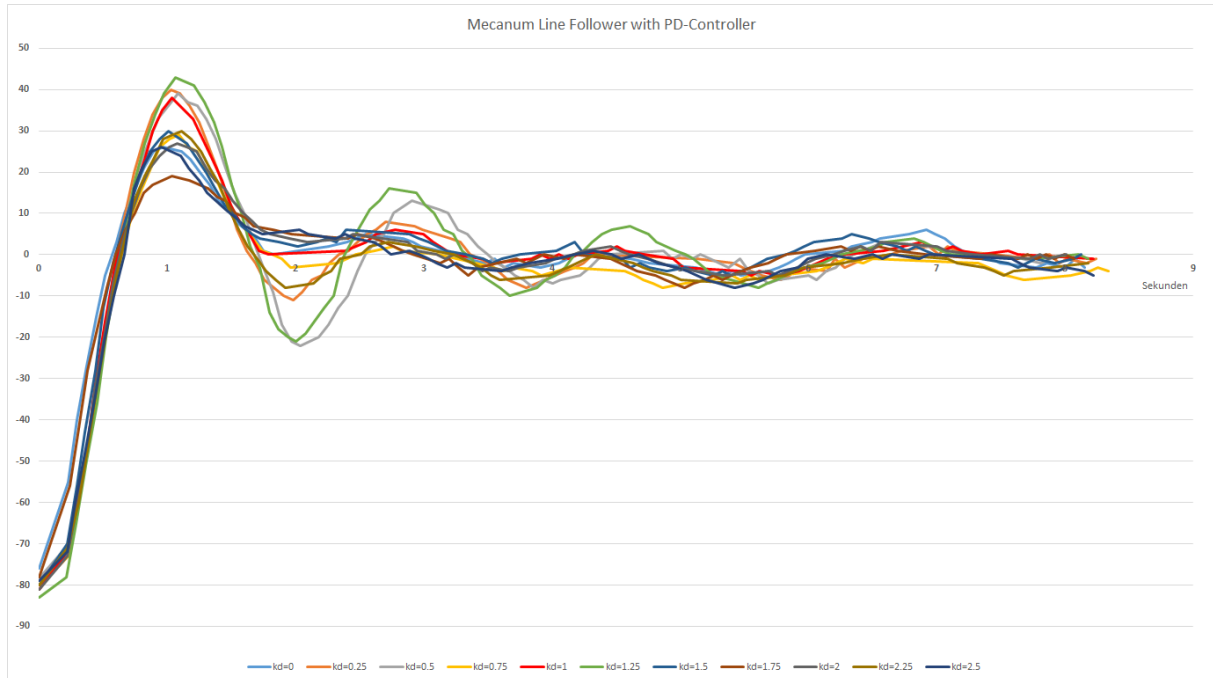
Program (example):

```

program start
  set speed to 500
  set kp to 6
  set kd to 1.75
  set deviation to 0
  set last_deviation to deviation
  wait until deviation ≠ 0
  set start_time to timestamp ms
  forward with:
    velocity speed
  repeat forever
  do set change to deviation - last_deviation
  set last_deviation to deviation
  + if change ≠ 0
  do log_data
  set speed_alignment to round with 0 decimals deviation × kp
  + round with 0 decimals change × kd
  + if speed_alignment > 0
  do turn with:
    velocity_left speed
    velocity_right speed
    - min of list + - create list with speed_alignment
    speed
  else if speed_alignment < 0
  do turn with:
    velocity_left speed
    + max of list + - create list with speed_alignment
    - speed
    velocity_right speed
  else forward with:
    velocity speed
  wait ms 10
  
```

Mecanum_Line_Follower_with_PD_Controller.ft

Measurement results for the PD controller (with $k_d \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5\}$):



Mecanum_Line_Follower_with_PD_Controller_Results.jpg

The differential factor $k_d = 1.75$ was most effective in dampening overshoot in the completed tests. The value can differ depending on the position of the line detection in the camera image and small structural differences between models.

Annex

Task 2: Track follower

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection to transmit the program to the TXT4.0.
- Course sheet with straight black line 2 cm wide
- Course sheet with black line in a closed circle, 2 cm wide (from the Robotics TXT 4.0 Base Set)
- Obstacle (box, can, ...)

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](#). github.io
- [2] Wikipedia: [Finite automaton \(state machine\)](#)
- [3] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme: [Modeling Software with Finite State Machines. A Practical Approach](#). Auerbach Publications, 2006.
- [4] Online diagram editor for creating state diagrams (Format drawio): <https://www.diagrammeditor.de/>
- [5] Wikipedia: [Control technology](#).
- [6] Wikipedia: [Controller](#).
- [7] RN-Wissen: [Control technology](#).
- [8] Tim Wescott: [PID without a PhD](#). Embedded Systems Programming, 10/2000, p. 86-108.

Name: _____

Class: _____

Date: _____

Task 4

Drawing robot

The Mecanum-Omnwheels vehicle is now equipped with a pen – turning it into a drawing robot.

Construction task

Construct the drawing robot according to the manual or convert the Mecanum-Omnwheels vehicle from 1, 2 or 3 into a drawing robot. Connect the encoder motors and the servomotor as specified on the wiring diagram.

Use the interface test or your control program from task 1 to check whether all the motors have been connected properly.

Important: Fix the pen (fineliner, felt tip) in the bracket in such a way that the tip is resting on the paper. Then start the TXT. The servo is automatically set to the middle setting. Then fit the servo lever onto the servo so that it is pointing forwards about horizontally and just about does not lift the pen with the pen holder.

Use the interface test to test lowering of the pen via the servo.

Attention: If you move the servo by hand with the TXT switched on, you can damage it.

Programming tasks

1. Lowering the pen

For the drawing robot to be able to move without drawing a line, the pen has to be lifted by the servo motor.

Add a function for lifting and lowering the pen to your function library for the Mecanum-Omnwheels vehicle. Select the servo position with the aid of the interface test.

2. “House of Santa Claus”

Using the navigation functions from task 1, you can now make the drawing robot draw the “House of Santa Claus” without lifting the pen from the paper.

At the end, the drawing robot should lift the pen and move to the side a little.

3. n-sided polygon

Now the robot is to draw any n-sided polygon with a fixed edge length (as in task 7 of the Robotics TXT 4.0 Base Set). Try to design the program as generally and – using loops – as compactly as possible.

Test the program by having the drawing robot draw a triangle, a rectangle, a pentagon ... and finally a 15-gon one after the other.

Tip: Do not choose an edge length that is too big.

Experimental tasks

1. Approach target point

The drawing robot should now move from its current position to a (target) point defined by coordinates (x, y) . Assume for this that the pen is at the zero point of the coordinate reference system $(0, 0)$ when the program starts and the robot is aligned along the (positive) X-axis.

1a. What movements does the drawing robot have to complete to travel from its position (zero point) to a given point (x, y) by the shortest route? What calculations are necessary for this?

Illustrate your thoughts with a drawing.

1b. Using the navigation functions from task 1, write a program that makes the drawing robot move to the given point (x, y) by the shortest route. Test your program with points measured beforehand.

2. “Draw by numbers”

In the program template “*Mecanum_Drawing_Coordinates.ft*” you will find two lists with x and y coordinates as well as one with “up/down” flags. The x and y coordinates describe one point each in the coordinate system with a point in the bottom left-hand corner of the paper as the starting point $(0, 0)$.

Position the drawing robot in such a way that the pen stops directly above the point $(0, 0)$ and align the robot facing right, parallel to the bottom edge of the paper. The drawing paper should be at least 60 cm wide and 40 cm high.

2a. Expand the control program for the drawing robot from experimental task 1 in such a way that the points (coordinates) on the list are approached one after the other. If the

“up/down” flag belonging to the coordinate is equal 0, the drawing robot is to travel with the pen lifted, if it is equal to 1, it is to draw a line to this point. You can set the size of the picture by multiplying the coordinates by a fixed factor.

2b. Take a photo of the picture the robot has drawn. What does it show?

2c. Now you can prescribe the drawing robot your own coordinate lists for a picture.

Attachments

Task 4: Drawing robot

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection for transmitting the program to the TXT4.0.
- Pen (fineliner, felt pen), large sheet of white paper
- Program template “*Mecanum_Drawing_Coordinates.ft*”

Further information

- [1] Look for “Coordinate Grid Picture” on the internet.
- [2] Oliver Boorman: [Cartesian Grid Image Generator](#).

Task 4: Drawing robot

The Mecanum-Omnwheels vehicle now becomes a drawing robot. It learns to draw geometric shapes according to specification using a felt tip or fineliner and it is taught to "draw by numbers".

The task follows on from task 7 of the Robotics TXT 4.0 Base Set.

Topic

Control of the Mecanum-Omnwheels vehicle to draw geometric shapes and prescribed lines.

Learning objectives

- Use of functions for clear program design
- Modelling and calculation of vehicle control (trigonometry)
- Design of data structures

Time required

For conversion of the Mecanum-Omnwheels vehicle from task 1 and 2 into a drawing robot, pupils need approx. 45 minutes; in the case of a complete reconstruction according to construction manual up to 75 minutes (assuming experience with fischertechnik).

For development of the control program to solve programming tasks, pupils need about 90 minutes, assuming previous knowledge from the Robotics TXT 4.0 Base Set (especially task 7). The time needed for solving the experimental tasks – depending on age and experience – is 135-180 minutes.

Attachments

Task 4: Drawing robot

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection for transmitting the program to the TXT4.0.
- Pen (fineliner, felt pen), large sheet of white paper
- Program template “*Mecanum_Drawing_Coordinates.ft*”

Further information

- [1] Look for “Coordinate Grid Picture” on the internet.
- [2] Oliver Boorman: [Cartesian Grid Image Generator](#).

Name: _____

Class: _____

Date: _____

Solution sheet task 4:

Drawing robot

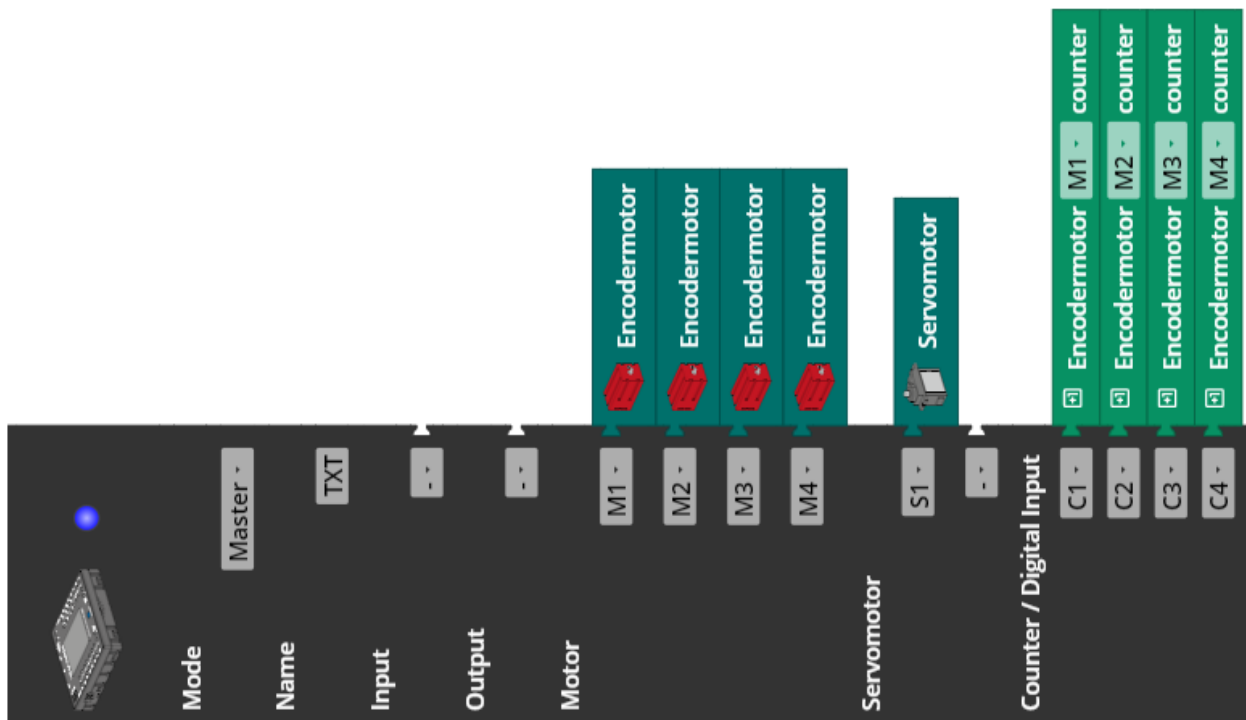
The tasks are an exercise in using loops and functions with the aim of getting a clear, compact and comprehensible program. For this reason, the pupils' results should be compared with one another.

Construction task

See building instructions.

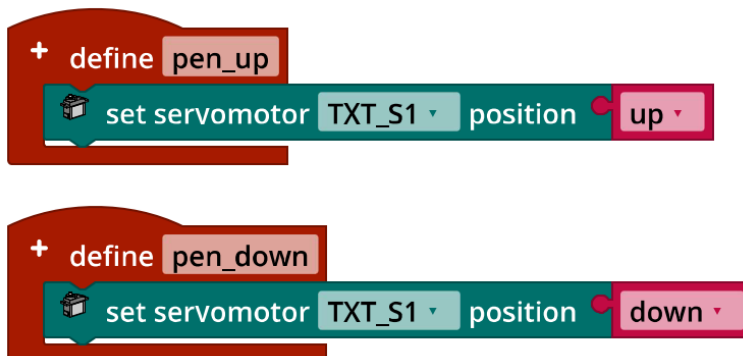
Programming tasks

Connection of the actuators:



1. Lowering the pen

Program excerpt (example):



The values for the “up” and “down” variables can be determined using the interface test. They are model-dependent and depend particularly on the position of the fitted servo arm.

2. “House of Santa Claus”

The “House of Santa Claus” can be drawn with eight lines without taking the pen off the paper. There are in fact 88 different correct possibilities.

The following program example draws it with an edge length of 30 cm and uses the functions “forward_distance” and “pivot_left_angle” from task 1.

Program (example):

```

program start
  set speed to 450
  set impulses_per_degree to 1.7
  set impulses_per_cm_straighton to 6.438
  set edge to 30
  set up to 280
  set down to 240
  pen_down

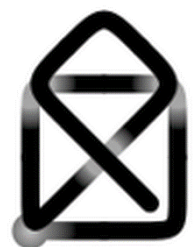
  repeat 4 times
    do forward_distance with:
      velocity speed
      distance edge
    pivot_left_angle with:
      velocity speed
      angle 90
  pivot_left_angle with:
    velocity speed
    angle 45
  forward_distance with:
    velocity speed
    distance edge * sqrt(2)
  pivot_left_angle with:
    velocity speed
    angle 90
  repeat 2 times
    do forward_distance with:
      velocity speed
      distance edge * sqrt(2) + 2
    pivot_left_angle with:
      velocity speed
      angle 90
  forward_distance with:
    velocity speed
    distance edge * sqrt(2)
  pivot_left_angle with:
    velocity speed
    angle 45
  pen_up
  forward_distance with:
    velocity speed
    distance edge * 2

+ define forward_distance with:
  - variable: velocity
  - variable: distance
  + - set motor TXT_M_M1 ccw speed velocity
    step size round with 0 decimals distance
    x impulses_per_cm_straighton
  sync with motor TXT_M_M2 direction ccw
  sync with motor TXT_M_M3 direction ccw
  sync with motor TXT_M_M4 direction ccw
  wait until
    and has motor TXT_M_M1 reached position
    and has motor TXT_M_M2 reached position
    and has motor TXT_M_M3 reached position
    and has motor TXT_M_M4 reached position

+ define pivot_left_angle with:
  - variable: velocity
  - variable: angle
  + - set motor TXT_M_M1 cw speed velocity
    step size round with 0 decimals angle
    x impulses_per_degree
  sync with motor TXT_M_M2 direction ccw
  sync with motor TXT_M_M3 direction cw
  sync with motor TXT_M_M4 direction ccw
  wait until
    and has motor TXT_M_M1 reached position
    and has motor TXT_M_M2 reached position
    and has motor TXT_M_M3 reached position
    and has motor TXT_M_M4 reached position

+ define pen_up
  set servomotor TXT_M_S1 position up

+ define pen_down
  set servomotor TXT_M_S1 position down
  
```



Mecanum_House_of_Santa_Claus.ft

3. n-sided polygon

The sum of the interior angles of an n-sided polygon is $(n - 1) \cdot 180^\circ$. To be able to draw an n-sided polygon, the Mecanum-Omnwheels vehicle must thus be turned through $180^\circ - \frac{(n-1) \cdot 180^\circ}{n}$ after every edge. The example solution generalises the task for drawing n-sided polygons and first draws a triangle, then a rectangle up to a 15-gon with an edge length of 20 cm each. Two nested loops make the program very compact.

Program excerpt (example):

```

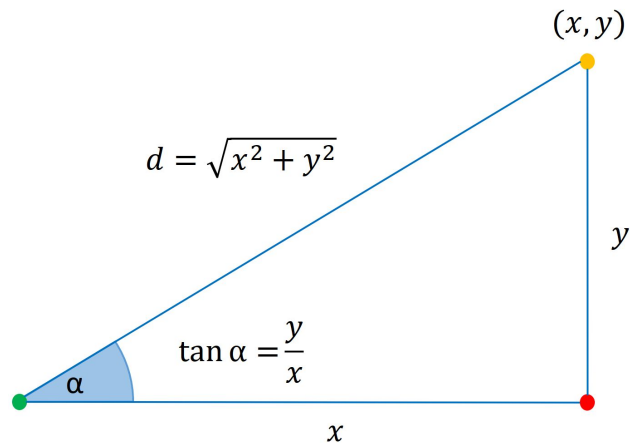
program start
  set speed to 450
  set impulses_per_degree to 1.7
  set impulses_per_cm_straighton to 6.438
  set edge to 20
  set up to 280
  set down to 240
  set angles to 2
  pen_down
  repeat 13 times
    do change angles by 1
    repeat angles times
      do forward_distance with:
         velocity speed
         distance edge
      pivot_left_angle with:
         velocity speed
         angle 180 - angles * 180 / 2
    pen_up
  forward_distance with:
    velocity speed
    distance edge * 2
  
```

Mecanum_Drawing_Polygons.ft

Experimental tasks

1. Approach target point

1a. To get from the point $(0, 0)$ to a point (x, y) , the drawing robot must move along the hypotenuse of a right-angled triangle with the leg lengths x and y .



Mathematical_Model_Coordinates_Drawing.jpg

The length of the hypotenuse is calculated using the Pythagoras' theorem:

$$d = \sqrt{x^2 + y^2}$$

The interior angle α of the right-angled triangle can be determined using a little trigonometry:

$$\tan \alpha = \frac{y}{x}$$

The angle or rotation δ – in relation to the X-axis – must be derived from this. The easiest way to do this is with a case distinction:

- As long as $x > 0$: $\delta = \alpha$
- If $x < 0$: $\delta = 180^\circ + \alpha$

The case $x = 0$ must be treated separately so there is no dividing by 0. For $y > 0$, $\delta = 90^\circ$ in this case, otherwise $\delta = -90^\circ$.

1b. Program (example):

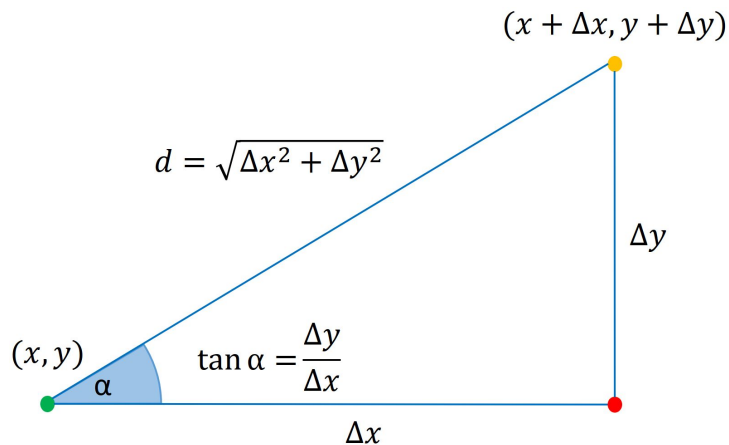
```

program start
set speed to 350
set impulses_per_degree to 1.7
set impulses_per_cm_straighton to 6.438
set delta to 0
set x to 25
set y to 35
+ if x = 0
do
+ if y < 0
do set delta to -90
else set delta to 90
else
set alpha to atan(y/x)
+ if x > 0
do set delta to alpha
else set delta to alpha + 180
+ if delta > 180
do set delta to delta - 360
+ if delta > 0
do pivot_left_angle with:
velocity speed
angle delta
else pivot_right_angle with:
velocity speed
angle absolute delta
set d to square root(square(x) + square(y))
forward_distance with:
velocity speed
distance d
    
```

Mecanum_Move_2_Point.ft

2. “Draw by numbers”

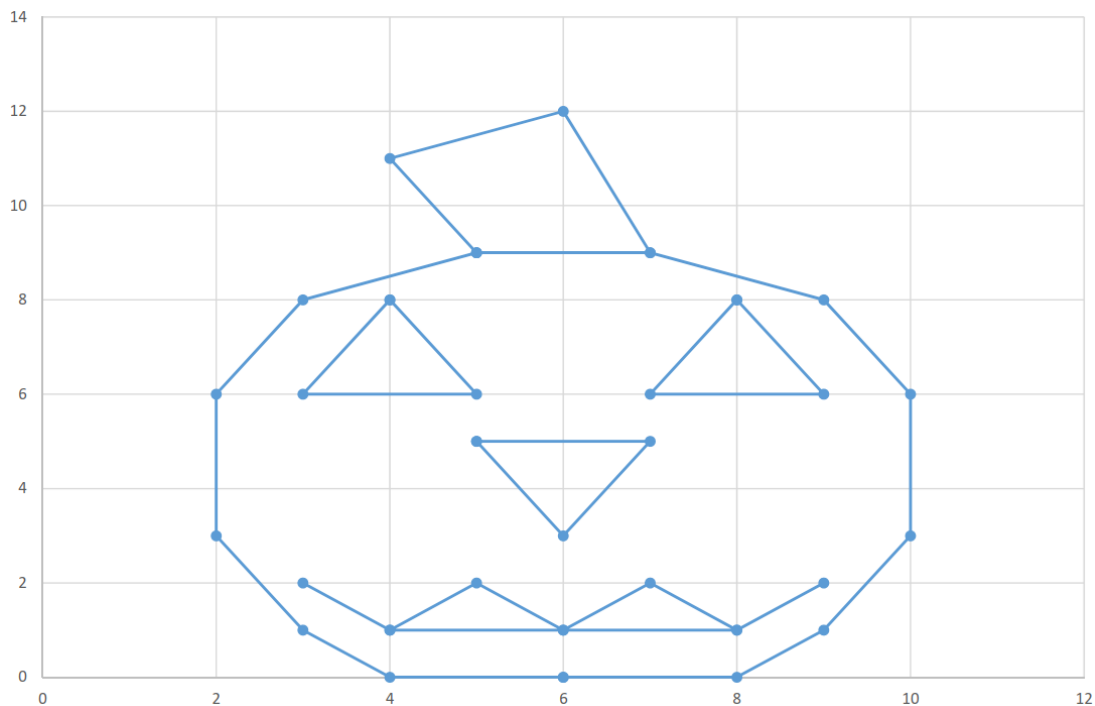
In the general case, the drawing robot is not aligned along the X-axis, rather it is at an angle γ to the X-axis; the angle of rotation δ must be set in relation to this angle. The distance to the (next) target point must be calculated in relation to the position of the robot $((\Delta x, \Delta y))$.



Mathematical_Model_Coordinates_Drawing_generalised.jpg

The points to be approached are saved in three lists with the respective x and y coordinates and an “up/down” flag which specifies whether the pen is to be lowered (1) or lifted (0) on the way to the point described by the coordinates.

The coordinates result in the following drawing:



Pumpkin.jpg

Program excerpt (example):

```

set index to 1
repeat length of coordinates_x times
do
set delta_x to in list coordinates_x get # index
- x
set x to in list coordinates_x get # index
set delta_y to in list coordinates_y get # index
- y
set y to in list coordinates_y get # index
+ if delta_x = 0
do
+ if delta_y < 0
do set delta to -90
else set delta to 90
else
set alpha to atan delta_y
+ delta_x
+ if delta_x > 0
do set delta to alpha
else set delta to alpha
+ 180
set next_angle to delta
set delta to delta
- gamma
+ if delta > 180
do set delta to delta
- 360
else if delta < -180
do set delta to delta
+ 360
set gamma to next_angle
+ if in list up_down get # index = 0
do pen_up
else pen_down
+ if delta > 0
do pivot_left_angle with:
velocity speed
angle delta
else pivot_right_angle with:
velocity speed
angle absolute delta
set d to square root square delta_x
+ square delta_y
x scale
forward_distance with:
velocity speed
distance d
change index by 1
    
```

Mecanum_Coordinates_Drawing.ft

Attachments

Task 4: Drawing robot

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection for transmitting the program to the TXT4.0.
- Pen (fineliner, felt pen), large sheet of white paper
- Program template "*Mecanum_Drawing_Coordinates.ft*"

Further information

- [1] Look for "Coordinate Grid Picture" on the internet.
- [2] Oliver Boorman: [Cartesian Grid Image Generator](#).

Name: _____

Class: _____

Date: _____

Task 5

Ball robot

In this task, the Mecanum-Omnwheels vehicle is equipped with a firing device for hollow plastic balls, receives voice control and is supplemented with autonomous target alignment with the help of a camera.

Construction task

Construct the ball robot according to instructions or convert the Mecanum-Omnwheels vehicle from one of the other tasks accordingly. Connect the encoder motors, the camera and the servomotor as specified on the wiring diagram.

Use the interface test or your control program from task 1 to check whether all the motors have been connected properly.

Important: Only fix the servo lever in place after you have started the TXT. The servo is automatically set to the middle setting. Then fit the servo lever onto the servo in such a way that it is pointing about “straight on” (in other words to the left in the model) and comes to rest behind the green firing strut.

Attention: If you move the servo lever by hand with the TXT switched on, the servo can undergo irreparable damage!

Programming tasks

1. Firing mechanism

Firing of a hollow plastic ball is triggered by moving the servo lever as far back as possible until it releases the tensioned green launching strut. During this, a hollow plastic ball slides automatically from the magazine into the firing position. Then the servo lever has to be moved back in front of the firing strut.

1a. Use the interface test to determine suitable positions for the servo lever for firing a hollow plastic ball and “preloading” the firing strut again.

1b. Add a function for firing a hollow plastic ball to your function library for the Mecanum-Omnwheels vehicle.

1c. Fix a button on the side of your vehicle and connect it to I1. Write a Blockly program that will fire a hollow plastic ball when the button is pressed.

1d. Extend your program by magazine fill level indication on the TXT display. Request reloading when all the hollow plastic balls have been fired and have successful reloading confirmed by button.

2. Voice control

You can also control your Mecanum-Omnwheels vehicle using voice commands. To do so, download the “Voice Control” app from the Apple app store (for iOS) or the Google Play store (for Android), and connect it to the TXT 4.0.

- Connection via WiFi: The TXT 4.0 Controller and device (smartphone or tablet) must be connected to the same WiFi router. The router must also permit communication between the devices. The IP address of the TXT 4.0 with which the app must be connected can then be queried via the touchscreen menu under “Info” / “WiFi”.
- Connection via WiFi AP: Instead of “WiFi”, the “Access Point” option can be activated on the TXT 4.0 under “Settings” / “Network”. Then the smartphone can be connected directly to the controller. The WPA2 key required for the WiFi connection is available in the TXT menu under “Access Point” (it can also be changed and deactivated there).

Once you have connected the app to the Controller, the voice commands are transmitted to the Controller in text form, and you can analyse them using the following event function:



In your program from programming task 1, replace the function of the button for triggering a shot by a suitable voice command that your smartphone will easily recognise.

Experimental tasks

In the following four tasks, the Mecanum-Omnwheels vehicle is equipped step-by-step with an automatic target finder.

For this, build a coloured target first in accordance with the construction drawing, and set it up at a distance of about 50 cm in front of your Mecanum-Omnwheels vehicle.

1. Target

1a. Activate the camera in the camera configuration. Configure ball detection in such a way, that the centre of the target is exactly in the centre of the detection window when the firing device hits the target as reliably as possible. Test the setting using your program from programming task 1.

Note: The camera is upside-down, which means the image must be rotated through 180° in the camera settings.

Camera settings

Resolution
320x240

FPS
15

Rotate image 180 degrees

CANCEL APPLY

1b. If you move the vehicle closer to the target or further away from it, the y-coordinate of ball detection changes. Enter the respective y-coordinate for the distance to the ideal position in the table below.

Distance to ideal position	y-coordinate
15 cm	
10 cm	
5 cm	
0 cm	
-5 cm	
-10 cm	
-15 cm	

1c. From this measurement, derive a simple approximation formula that you can use to calculate how many cm the Mecanum Omniwheels vehicle must move forward or backward to be at an ideal distance to the target.

2. Target distance correction

Write a Blockly program which makes the Mecanum-Omnwheels vehicle take up the exactly correct distance to the target. Use your results from experimental task 1 for this.

Sketch a state transition diagram first.

3. Target alignment

Write a Blockly program which rotates the Mecanum-Omnwheels vehicle in such a way that the target is exactly in the middle of the detection window.

Draw a state transition diagram for your solution first.

Tip: The camera has an aperture of 60° .

4. Target finder

Now your solutions developed in the previous sub-tasks are to be combined. Write a Blockly program which makes the Mecanum-Omnwheels vehicle find a target first. Once it has found it, it should move the target exactly to the centre of the crosshairs and hit the target with the three hollow plastic balls from the magazine.

First illustrate your solution concept with a state transition diagram.

Finally, you can let the TXT 4.0 play a sound.

.

Attachments

Task 5: Ball robot

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection for transmitting the program to the TXT4.0.
- Polystyrene balls

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](#). github.io
- [2] Wikipedia: [Finite state machine](#)
- [3] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme: [Modeling Software with Finite State Machines. A Practical Approach](#). Auerbach Publications, 2006.
- [4] Online diagram editor for creating state transition diagrams (drawio format): <https://www.diagrammeditor.de/>

Task 5: Ball robot

The Mecanum-Omnwheels vehicle receives a firing mechanism for polystyrene balls, voice control and a camera for independent target alignment.

Topic

Image evaluation for determining distance and target alignment.

Learning objectives

- Experimental analysis of a task
- “Computational Thinking”: Breaking down a complex task into manageable and solvable sub-tasks which are then brought together (“divide and rule” principle)

Time required

For conversion of the Mecanum-Omnwheels vehicle from previous tasks into a ball robot, pupils need approx. 90 minutes; in the case of a complete reconstruction according to construction manual up to 120 minutes (assuming experience with fischertechnik).

For the development of a control program for solving programming tasks, pupils require 60-90 minutes. The time needed for solving the experimental tasks – depending on age and experience – is 135-240 minutes.

Attachments

Task 5: Ball robot

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection for transmitting the program to the TXT4.0.
- Polystyrene balls

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](#). github.io
- [2] Wikipedia: [Finite state machine](#)
- [3] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme: [Modeling Software with Finite State Machines. A Practical Approach](#). Auerbach Publications, 2006.
- [4] Online diagram editor for creating state transition diagrams (drawio format): <https://www.diagrammeditor.de/>

Name: _____

Class: _____

Date: _____

Solution sheet task 5:

Ball robot

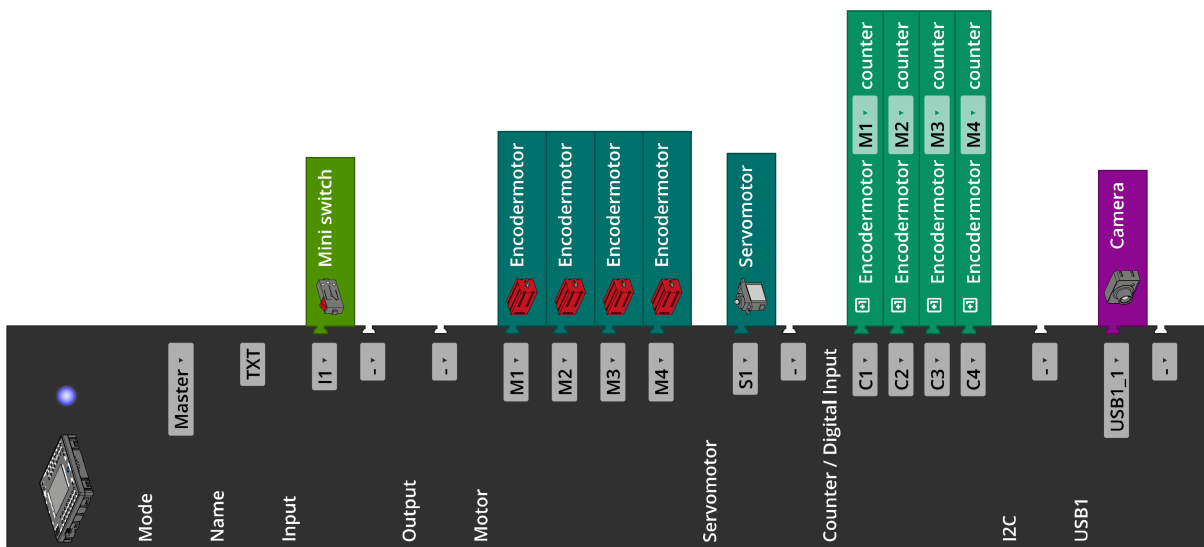
The tasks are an exercise in using loops and functions with the aim of getting a clear, compact and comprehensible program. For this reason, the pupils' results should be compared with one another.

Construction task

See building instructions.

Programming tasks

Configuring the sensors and actuators:



The “Voice Control” app (for iOS or Android) is required to solve programming task 2. The app must be connected to the internet for voice recognition, and connected to the Controller (via Bluetooth or WiFi).

1. Firing mechanism

1a. Suitable values for the variables “fire” and “load” (the positions of the servo lever looked for) can easily be determined using the interface test. They are model-dependent and depend particularly on the position of the fitted servo lever. In the model used here, they are between 120 (fire) and 380 (load).

1b. “Fire” function (example):

```

+ define fire
  set servomotor TXT_M_S1 position fire
  wait ms 250
  set servomotor TXT_M_S1 position load
  
```

Mecanum_Fire_and_Load.ft

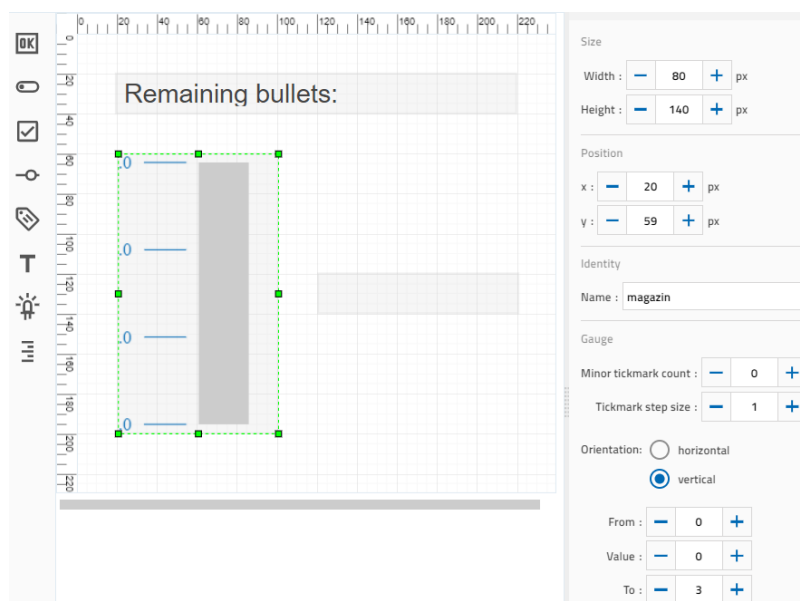
1c. Program excerpt (example):

```

program start
  set fire to 120
  set load to 380
  fire
  repeat forever
    do + if is mini switch TXT_M_I1 closed
      do fire
  
```

Mecanum_Fire_and_Load.ft

1d. Configuration of the TXT display (example):



Program (example):

Mecanum_Fire_and_Load_extended.ft

2. Voice control

Program (example):

Mecanum_Fire_and_Load_Voice_Command.ft

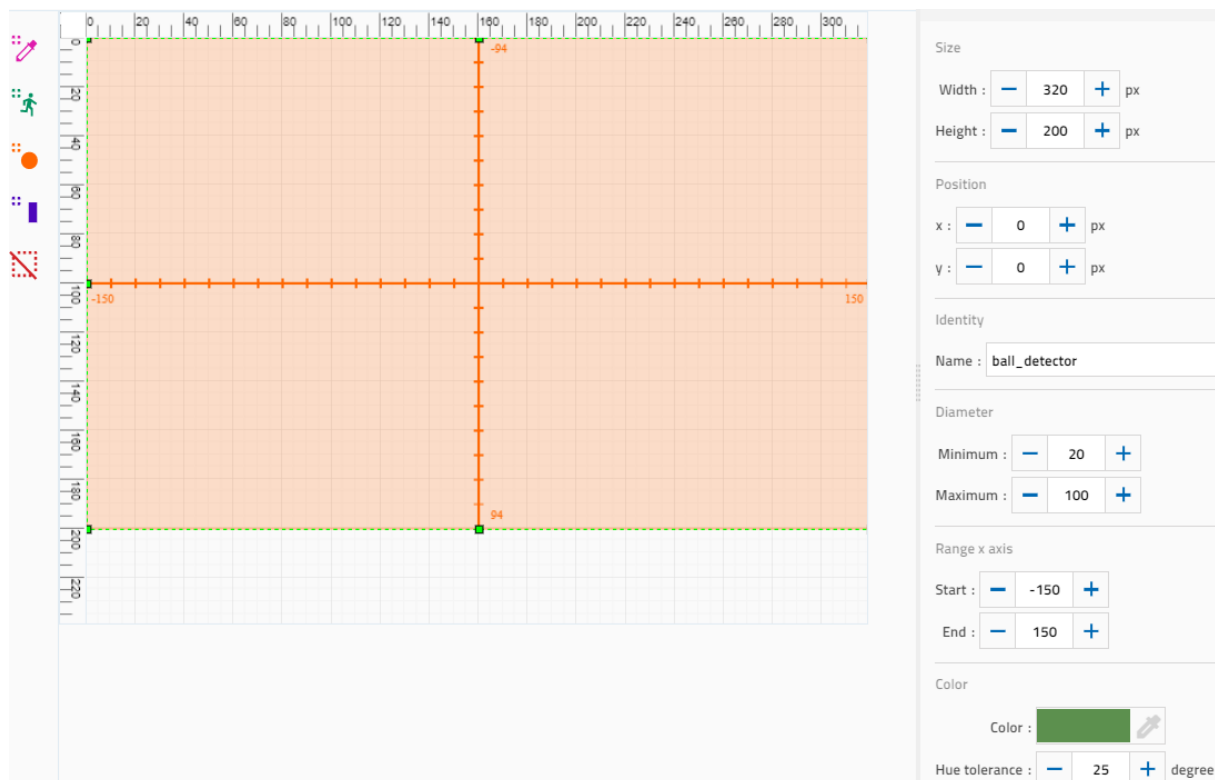
Experimental tasks

1. Target

1a. The target should ideally be at such a distance that the hollow plastic ball hits the target a little above the centre.

In the example model used, this ideal distance is around 35 cm. This model can vary slightly from model to model.

Configuration of the detection window (ball detection):



The larger the detection window and the area of the ball diameter to be detected, the larger the maximum deviation from the ideal distance that can be detected. The interval $[-150, 150]$ is to be recommended for the X-axis: It uses almost the complete resolution (320 pixel) and can easily be converted to an angle, since the camera aperture is 60° .

The colour and hue tolerance selected must be adapted to the respective light conditions.

1b. Program for distance measurement (example):

```

program start
  set y to 0
  repeat forever
    do
      set last_y to y
      set ball_detected to 0
      wait until ball_detected = 1
      + if last_y ≠ y
      do log_data

on ball ball_detector detected: event
  set y to get y-position of ball event
  set ball_detected to 1

+ define log_data
  set label target_Distance text + - create text with "Target-Distance: "
  y
  
```

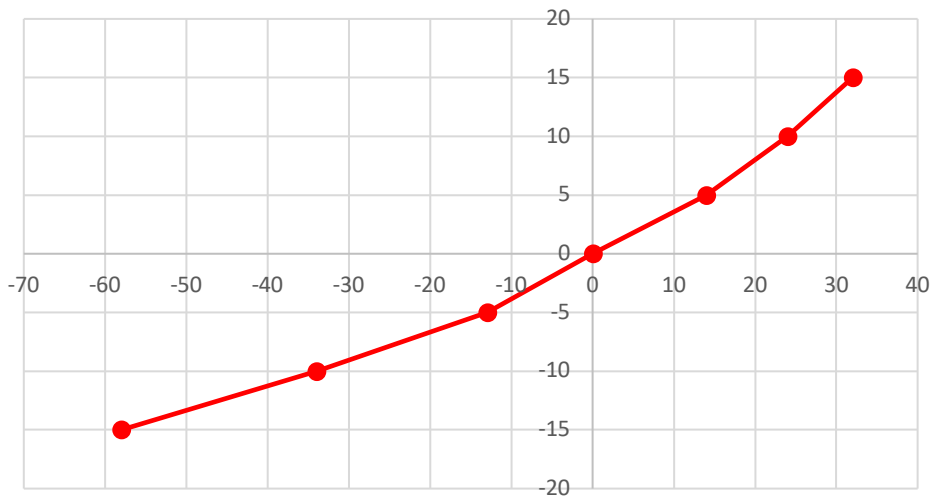
Test_Target_Distance.ft

Example measurement for the relation between the distance of the target and the y-value of the ball centre which is provided by ball detection:

Distance to ideal position	y-coordinate
+15 cm	32
+10 cm	24
+5 cm	14
0 cm	0
-5 cm	-13
-10 cm	-34
-15 cm	-58

The values of the y-coordinate of the ball centre returned by the ball detection depend on the scaling selected and the height of the detection window. However, except for a scaling factor, the values should match those given in the table above.

Distance to the ideal position



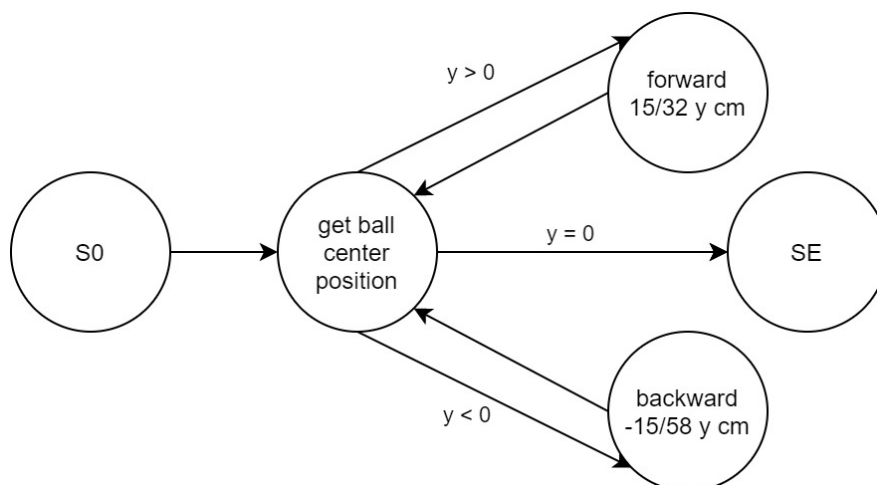
1c. The distance d by which the vehicle is away from the ideal distance to the target can easily be approximated using two linear functions:

$$d = \begin{cases} y > 0: \frac{15}{32}y \\ y = 0: 0 \text{ cm} \\ y < 0: \frac{15}{58}y \end{cases}$$

Again, the gradient factor of the straight line depends on the height of the detection window and the scaling set in the ball detection and may differ by a factor.

2. Target distance correction

2a. State transition diagram



State_Transition_Diagram_Correct_Position.drawio

2b. Program excerpt (example):

```

program start
  set speed to 350
  set impulses_per_cm_straighton to 6.82
  set ball_detected to 0
  wait until ball_detected = 1
  repeat while y ≠ 0
  do log_data
  + if y < 0
  do backward_distance with:
    velocity speed
    distance absolute round with 0 decimals (y * 15) / 58
  else forward_distance with:
    velocity speed
    distance round with 0 decimals (y * 15) / 32
  set ball_detected to 0
  wait until ball_detected = 1

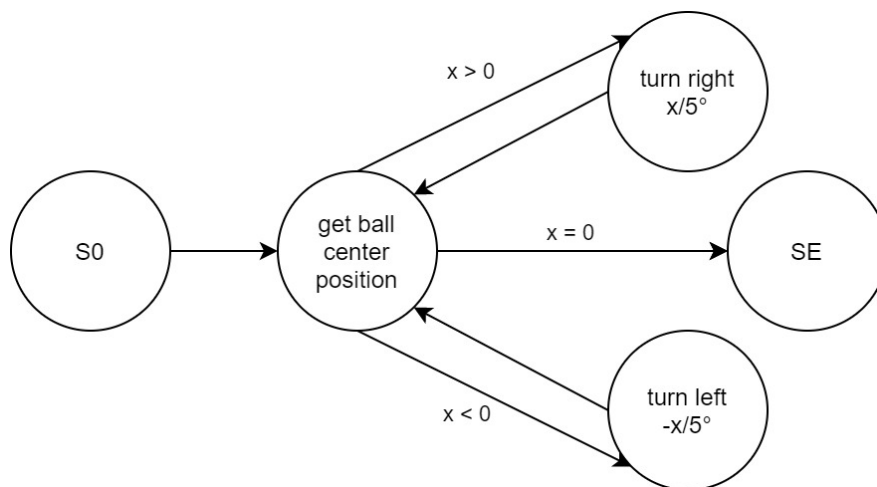
on ball ball_detector detected: event
  set y to get y-position of ball event
  set ball_detected to 1

+ define log_data
  print y
  
```

Mecanum_Correct_Distance.ft

3. Target alignment

3a. State transition diagram



State_Transition_Diagram_Correct_Position.drawio

3b. Program excerpt (example):


```

program start
  set speed to 350
  set impulses_per_degree to 1.7
  set ball_detected to 0
  wait until ball_detected = 1
  repeat while x ≠ 0
  do
    set x to x ÷ 5
    log_data
    + if x < 0
    do
      pivot_left_angle with:
        velocity speed
        angle absolute x
    else
      pivot_right_angle with:
        velocity speed
        angle x
    set ball_detected to 0
    wait until ball_detected = 1

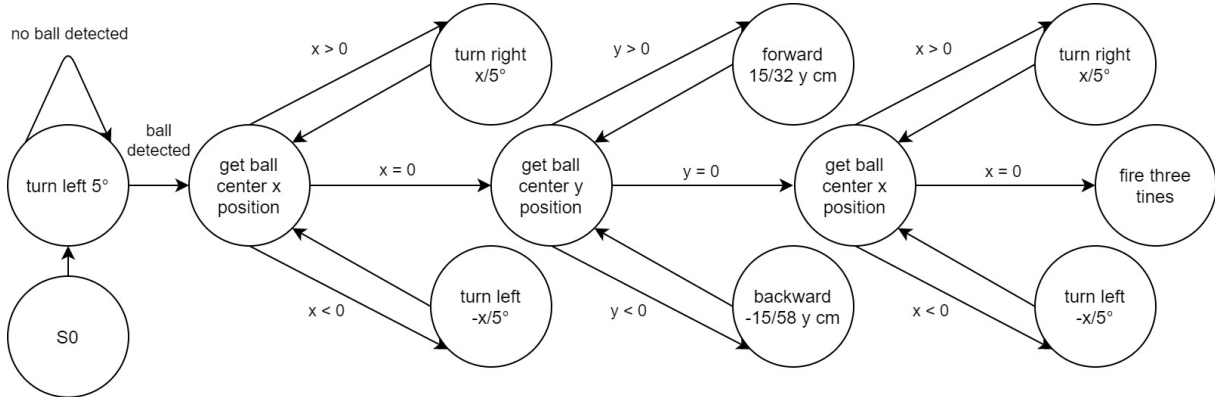
on ball ball_detector detected: event
  set x to get x - position of ball event
  set ball_detected to 1

+ define log_data
  print x
  
```

Mecanum_Turn2Target.ft

4. Target finder

4a. State transition diagram



State-Transition_Diagram_Find_Target.drawio

4b. Program excerpt (example):

```

program start
  set speed to 350
  set impulses_per_cm_straighton to 6.82
  set impulses_per_degree to 1.7
  set ball_detected to 0
  set fire to 120
  set load to 380
  fire
  repeat while ball_detected = 0
  do pivot_left_angle with:
    velocity speed
    angle 5
    wait ms 100
  repeat while x != 0
  do set x to x + 5
  + if x < 0
  do pivot_left_angle with:
    velocity speed
    angle absolute x
  else pivot_right_angle with:
    velocity speed
    angle x
  wait ms 200
  set ball_detected to 0
  wait until ball_detected = 1
  repeat while y != 0
  do + if y < 0
  do backward_distance with:
    velocity speed
    distance absolute y
    + 58
  else forward_distance with:
    velocity speed
    distance y
    + 32
  wait ms 200
  set ball_detected to 0
  wait until ball_detected = 1
  repeat while x != 0
  do set x to x + 5
  + if x < 0
  do pivot_left_angle with:
    velocity speed
    angle absolute x
  else pivot_right_angle with:
    velocity speed
    angle x
  wait ms 200
  set ball_detected to 0
  wait until ball_detected = 1
  wait ms 200
  11_Fantasy_3.wav start playing audio file
  repeat 3 times
  do fire
  wait ms 300
  wait until not is playing sound
  on ball ball_detector detected: event
  set x to get x-position of ball event
  set y to get y-position of ball event
  set ball_detected to 1
  
```

Mecanum_Find_Target.ft

Attachments

Task 5: Ball robot

Required materials

- PC for program development, local or via web interface.
- USB cable or BLE or WiFi connection for transmitting the program to the TXT4.0.
- Polystyrene balls

Further information

- [1] FRC Team 2605 (Bellingham, WA): [How a Mecanum Drive Works](https://github.io). github.io
- [2] Wikipedia: [Finite state machine](#)
- [3] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Wolstenholme: [Modeling Software with Finite State Machines. A Practical Approach](#). Auerbach Publications, 2006.
- [4] Online diagram editor for creating state transition diagrams (drawio format): <https://www.diagrammeditor.de/>