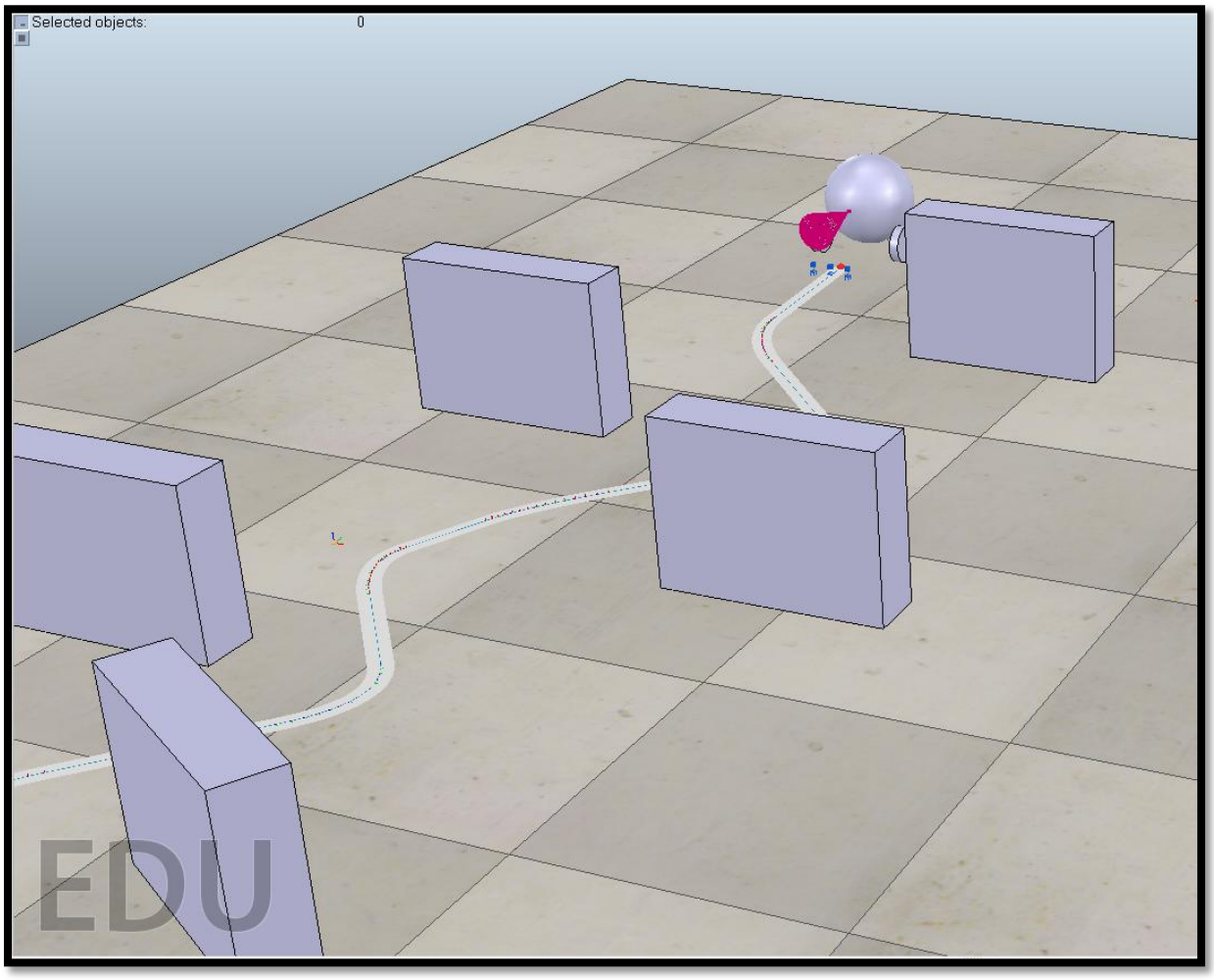
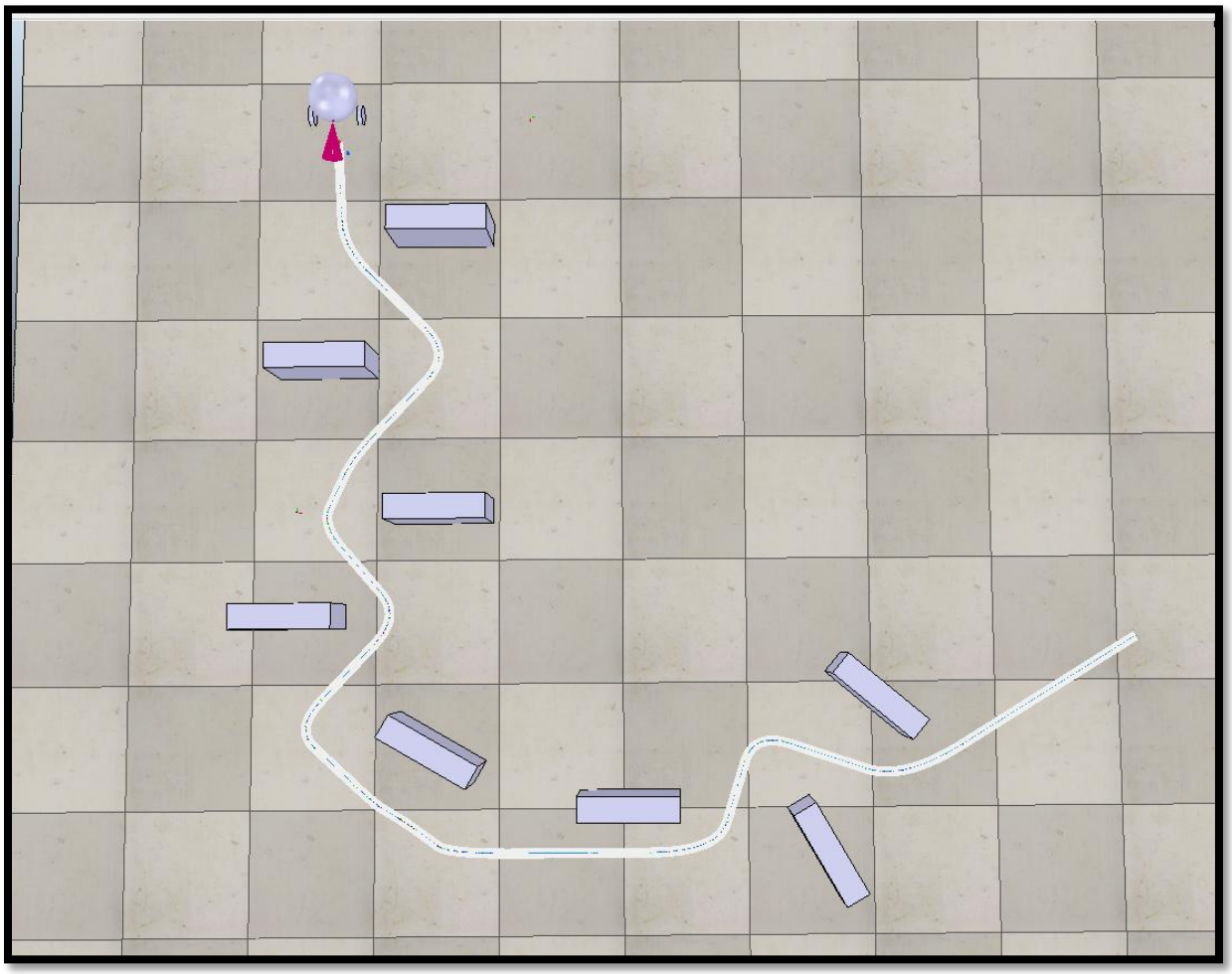
# Sprawozdanie Lab 6

Andrzej Żaba, gr_lab 4, nr indeksu: 401490

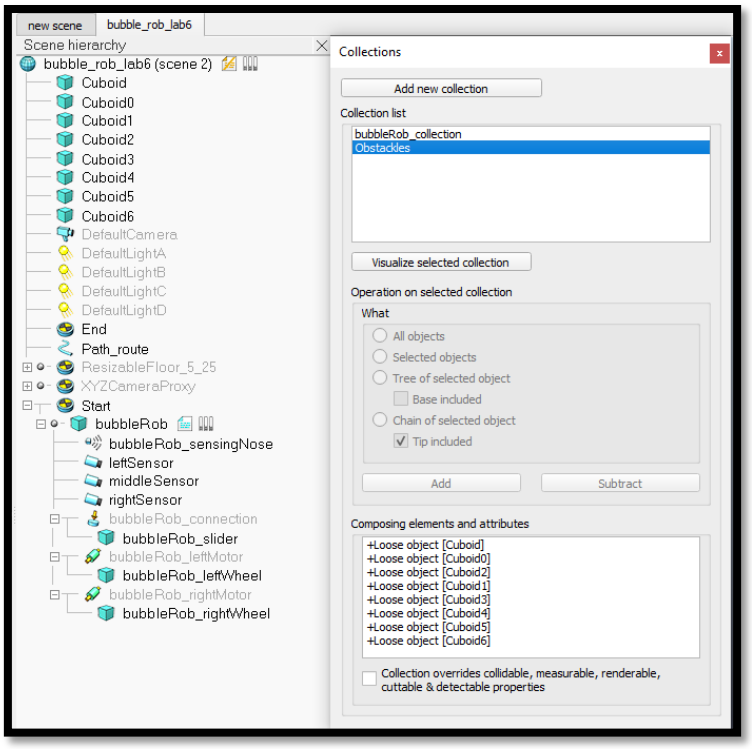
Zgodnie z instrukcją zaimportowano robota zrobionego na poprzednich zajęciach. Usunięto z niego trasę z punktami do zatrzymania się. Dodano do sceny przeszkody.



Z powodu braku odpowiednich funkcji (path planning) w obecnej wersji programu nie udało się zastosować automatycznego generowania trasy. Stworzono zatem przykładową trasę po której mógłby poruszać się robot omijając przeszkody w drodze do celu.



Wszystkim przeszkodom nadano właściwość kolizyjności. Utworzono z nich również kolekcję:




Nie odnaleziono również w programie możliwości poruszania się po ścieżce za pomocą ustawień ruchu. Postanowiono zrealizować ruch po ścieżce z użyciem sterownika napisanego w skrypcie.

Zastąpiono napisany na poprzednich zajęciach skrypt do jazdy po linii w danym kolorze skryptem typu **threaded child script.**

Na początku funkcji *function sysCall_threadmain()* zadeklarowano zmienne potrzebne w dalszej części pisania kodu:

```
-- Put some initialization code here

    -- motor assignment
    left_motor=sim.getObjectHandle("bubbleRob_leftMotor") -- Handle of the left motor
    right_motor=sim.getObjectHandle("bubbleRob_rightMotor") -- Handle of the right motor

    -- path assignment
    robot_handle = sim.getObjectHandle('bubbleRob')
    path_handle = sim.getObjectHandle('Path_route')

    position_on_path = 0
```

```
distance = 0    -- distance form the robot to the point on the path

start_dummy_handle = sim.getObjectHandle('Start')


-- Valocity calculations
v_desired = 0.1
om_desired = 0.1
d = 0.2     -- wheels separation
r_w = 0.04  -- wheel radius

v_r = (v_desired + d*om_desired)
v_l = (v_desired - d*om_desired)

omega_right = v_r / r_w
omega_left = v_l / r_w
```

Do odpowiednich zmiennych zostały przypisane obiekty takie jak silniki napędzające koła, sam bubbleRob, ścieżka po której ma odbywać się ruch oraz dodana w między czasie kukła (dummy) ‘Start’. W drzewie hierarchii podpięto naszego robota pod kukłę ‘Start’. Zastosowano również wzory potrzebne do ustalenia odpowiednich prędkości silników. Zadeklarowano również wstępnie pożądane do osiągnięcia prędkości liniową i obrotową oraz stałe wartości rozstawu kół i ich promienia.

Następnie napisano kod w części głównej skryptu – cyklicznie wykonującej się pętli.

```
-- While(1) loop
while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do

   -- set speed on both motors
   sim.setJointTargetVelocity(right_motor, omega_right)
   sim.setJointTargetVelocity(left_motor, omega_left)

   -- set current robot position
   robot_position = sim.getObjectPosition(robot_handle, -1)

   -- position of point on the path robot is going to
   path_position = sim.getPositionOnPath(path_handle, position_on_path)
```

Na początku następuje uruchomienie silników z zadaną prędkością. Odczytywana jest pozycja robota względem globalnego układu współrzędnych. Określana jest również pozycja punktu do którego robot ma zmierzać.

Dalej:

```
sim.setObjectPosition(start_dummy_handle,-1, path_position)

   m = sim.getObjectMatrix(robot_handle, -1)
   m = simGetInvertedMatrix(m)     -- if doesn't work try this:     sim.invertMatrix


   path_position = sim.multiplyVector(m,path_position)

   -- distance from robot to target point
   distance = math.sqrt( (path_position[1])^2 + (path_position[2])^2)    --
path_position[1] -> x coord   [2] -> y coord

   -- angle between robot heading and the path piont
   phi_angle = math.atan2(path_position[2], path_position[1])
```

Otrzymywana jest macierz transformacji robot – world. Kolejno jest odwracana w celu uzyskania transformacji z globalnego układu do lokalnego związanego z robotem.

Owa odwrócona macierz jest następnie mnożona z wcześniej odczytaną pozycją punktu do którego zmierza robot. Następnie określany jest dystans oraz kąt obrotu między robotem a punktem do którego zmierza. Użyte zostały odpowiednie wzory.

```
if position_on_path<1 then
    v_desired = 0.15
    om_desired = 1.5 * phi_angle
else
    v_desired = 0
    om_desired = 0
end


v_r = (v_desired + d*om_desired)
v_l = (v_desired - d*om_desired)

omega_right = v_r / r_w
omega_left = v_l / r_w
```

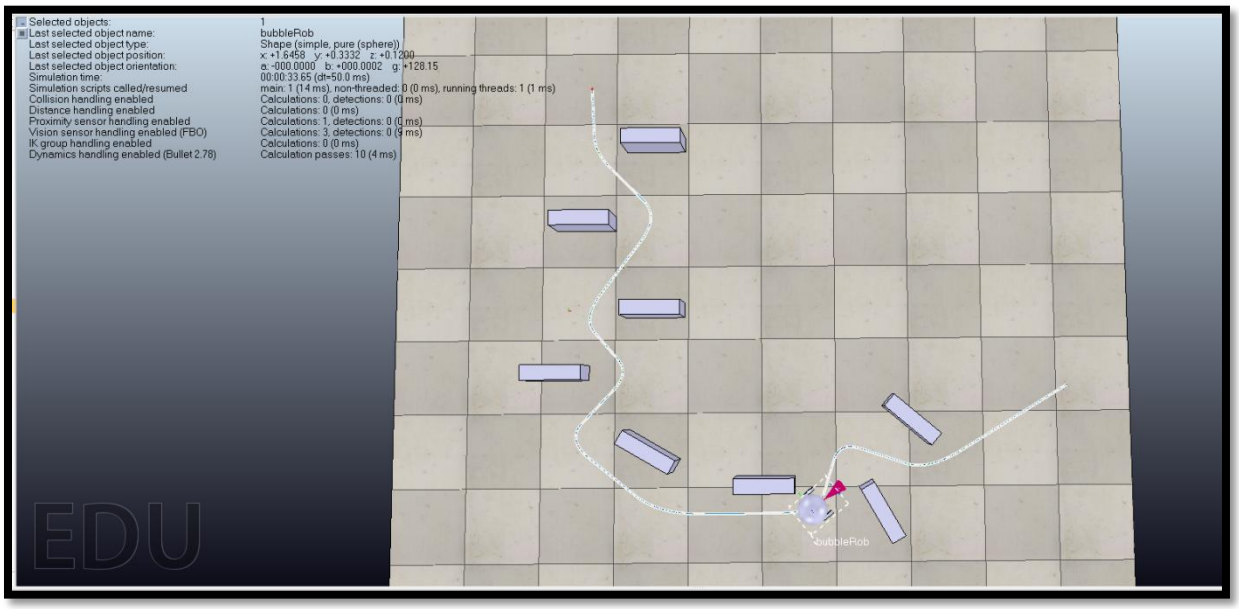
Następnie, jeśli jeszcze nie osiągnięto ostatniego punktu ścieżki aktualizowane są prędkości silników. Prędkość kątowa jest dobierana proporcjonalnie do kąta o jaki robot musi się obrócić aby być w linii prostej ze swoim celem. Jeśli natomiast robot przejechał ostatni punkt na ścieżce – ma się zatrzymać.

```
-- points on the path are numbered from 0 to 1.  So when distance from one
point is
-- really small -> we add 0.01 to the position_on_path untill position on path
-- would be equall the value of next point on the path

if(distance<0.1) then
    if(position_on_path<1) then
        position_on_path=position_on_path+0.01
    end
end

sim.wait(0.025, true)
```

Ostatnią rzeczą jest aktualizowanie punktu na ścieżce do którego dąży robot (przez dodanie 0.01). Drugi warunek został wprowadzony dodatkowo. Warunek zatrzymania się robota napisany w poprzedniej części nie sprawował się zadowalająco. Robot zatrzymywał się, jednak po pewnym czasie zaczynał jechać do przodu, krótkimi, ale agresywnymi ruchami. Dodatkowo wprowadzony warunek nie pozwala robotowi przekroczyć punktu o wartości 1 na ścieżce, przez co zatrzymuje się na jej końcu.




Dla wartości

```
if position_on_path<1 then
    v_desired = 0.15
    om_desired = 1.5 * phi_angle
  else
    v_desired = 0
    om_desired = 0
  end
```

Robot porusza się całkiem sprawnie i płynnie. Dobre rezultaty otrzymano też dla wartości 0.1 i 1.5. Zadowalający wynik – już dla szybszego trybu dają 0.18 i 1.85. Zdarza się wtedy lepszy lub gorszy przejazd, ale robot trzyma się trasy, kwestia płynności na zakrętach. Problematyczne staje się dobieranie wartości powyżej 0.2. Wtedy problemy na zakrętach pojawiają się bardzo często.

## Całość kodu:

```
function sysCall_threadmain()

 -- Put some initialization code here

   -- motor assignment
   left_motor=sim.getObjectHandle("bubbleRob_leftMotor") -- Handle of the left motor
   right_motor=sim.getObjectHandle("bubbleRob_rightMotor") -- Handle of the right motor

   -- path assignment
   robot_handle = sim.getObjectHandle('bubbleRob')
   path_handle = sim.getObjectHandle('Path_route')

   position_on_path = 0    -- determines to which point on the path robot should go
   distance = 0    -- distance form the robot to the point on the path

   start_dummy_handle = sim.getObjectHandle('Start')


   -- Valocity calculations
   v_desired = 0.1
   om_desired = 0.1
   d = 0.2     -- wheels separation
   r_w = 0.04  -- wheel radius

   v_r = (v_desired + d*om_desired)
   v_l = (v_desired - d*om_desired)

   omega_right = v_r / r_w
   omega_left = v_l / r_w




-- End of initialization code


-- While(1) loop
while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do

   -- set speed on both motors
   sim.setJointTargetVelocity(right_motor, omega_right)
   sim.setJointTargetVelocity(left_motor, omega_left)

   -- set current robot position
   robot_position = sim.getObjectPosition(robot_handle, -1)

   -- position of point on the path robot is going to
   path_position = sim.getPositionOnPath(path_handle, position_on_path)
```

```
    sim.setObjectPosition(start_dummy_handle,-1, path_position)

    m = sim.getObjectMatrix(robot_handle, -1)
    m = simGetInvertedMatrix(m)     -- try this if doesn't work:     sim.invertMatrix


    path_position = sim.multiplyVector(m,path_position)

    -- distance from robot to target point
    distance = math.sqrt( (path_position[1])^2 + (path_position[2])^2)    --path_position[1] -> x coord
[2] -> y coord

    -- angle between robot heading and the path piont
    phi_angle = math.atan2(path_position[2], path_position[1])


    if position_on_path<1 then
        v_desired = 0.15
        om_desired = 1.5 * phi_angle
    else
        v_desired = 0
        om_desired = 0
    end


    v_r = (v_desired + d*om_desired)
    v_l = (v_desired - d*om_desired)

    omega_right = v_r / r_w
    omega_left = v_l / r_w

    -- points on the path are numbered from 0 to 1.  So when distance from one point is
    -- really small -> we add 0.01 to the position_on_path untill position on path
    -- would be equall the value of next point on the path

    if(distance<0.1) then
        if(position_on_path<1) then
            position_on_path=position_on_path+0.01
        end
    end

    sim.wait(0.025, true)


end -- end while loop


    -- Put your main loop here, e.g.:
    --
    -- while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
    --     local p=sim.getObjectPosition(objHandle,-1)
    --     p[1]=p[1]+0.001
```

```
    --     sim.setObjectPosition(objHandle,-1,p)
    --     sim.switchThread() -- resume in next simulation step
    -- end


end -- end SysCall function (main())

function sysCall_cleanup()
    -- Put some clean-up code here
end
```