Chapter 1

Inleiding

Algorithm 1: Learning algorithm

Result: $\boldsymbol{\theta}_{opti}$
initialization:
$\boldsymbol{\theta} = [1, 1, ..., 1, 1]$
$\tilde{\boldsymbol{f}} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{f}(\boldsymbol{r}_{obs,i})$;
while *Not converged* **do**
  for i = *1...m* **do**
    $\min\limits_{r_{exp}} \quad \boldsymbol{\theta}^T \cdot \boldsymbol{f}(\boldsymbol{r}_{exp})$;
    constraints:
      $opti.begin = initial(observed_i)$;
      $opti.end = [y = lane\_distance_i, vy = 0, ay = 0, jy = 0]$;
  end
  $E_{p(\boldsymbol{r}|\boldsymbol{\theta})}(\boldsymbol{f}) = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{f}(\boldsymbol{r}_{exp,i})$;
  $\Delta\boldsymbol{\theta} = \alpha \cdot (E_{p(\boldsymbol{r}|\boldsymbol{\theta})}(\boldsymbol{f}) - \tilde{\boldsymbol{f}})$;
  $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$;
  if $\Delta\boldsymbol{\theta} \leq \epsilon$ **then**
    return $\boldsymbol{\theta}$;
  end
end

Algorithm 2: Double optimization

Result: $\boldsymbol{\theta}_{opti}$

initialization:

$\boldsymbol{\theta} = [\frac{1}{nf}, \frac{1}{nf}, ..., \frac{1}{nf}, \frac{1}{nf}]$, nf: number of features

$\tilde{\boldsymbol{f}} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{f}(\boldsymbol{r}_{obs,i})$;

while *Not converged* **do**

  start opti 1:

  $\boldsymbol{r}_{exp,i} = argmin(\boldsymbol{\theta}^T \cdot \boldsymbol{f}(\boldsymbol{r}_{exp,i})), \forall i \in 1 : m$;

  constraints:

    $opti_1.time = fixed\_time$;

    $opti_1.begin = initial(observed_i)$;

    $opti_1.end = [y = lane\_distance, vy = 0, ay = 0, jy = 0]$;

  end

  $E_{p(\boldsymbol{r}|\boldsymbol{\theta})}(\boldsymbol{f}) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{f}(\boldsymbol{r}_{exp,i})$;

  start opti 2:

  $\Delta\boldsymbol{\theta} = argmin \sum_{i=1}^{nf} -\Delta\theta_i \cdot (f_i - \tilde{f}_i)$;

  constraints:

    $opti_2.subject\_to(\Delta\theta_i \cdot (f_i - \tilde{f}_i) \geq 0)$;

    $opti_2.subject\_to(\sum_{i=1}^{nf} \Delta\theta_i == 0)$;

  end

  $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$;

  if $\Delta\boldsymbol{\theta} \leq \epsilon$ **then**

    $return(\boldsymbol{\theta})$;

  end

end

Figure 1.1: Dit is de sedes van de kul. Der staat trouwens bij dat de KUL in 1425 ontstaan is. Deze caption is uitzonderlijk lang om te testen ofdat het werkt om de caption over twee lijnen te zetten op 0.8 tekstbreedte.

Algorithm 3: Update theta

Result: $\boldsymbol{\theta}_{new}$

$\Delta\boldsymbol{\theta} = [0, 0...0]$;

for *i = [1...nf]* **do**

  if $\boldsymbol{f_{diff}}[i] > 0$ **then**

    $\Delta\boldsymbol{\theta}[i] = \Delta\boldsymbol{\theta}[i] + \boldsymbol{f_{diff}}[i]$

  else

    for *k = [1...nf]* **do**

      if *k!=i* **then**

        $\Delta\boldsymbol{\theta}[i] = \Delta\boldsymbol{\theta}[i] - \boldsymbol{f_{diff}}[i]$

      end

    end

  end

end

$\boldsymbol{\theta} = \frac{\boldsymbol{\theta}}{norm_2(\boldsymbol{\theta})}$

Algorithm 4: MPC loop

$total\_trajectory =$
$[[0, [punt_1x, punt_1y], [punt_2x, punt_2y]],$
$[1, [centre_x, centre_y], radius, start\_degree, end\_degree]];$
$current\_state = intitial\_state\_system;$
$N = 2;$
for *i in [0:1:len(total_trajectory)]* **do**
  $seed\_trajectory = total\_trajectory[i : i + N];$
  $[next\_inputs, calculated\_states, calculated\_controls] =$
   $\boldsymbol{inner\_optimization(seed\_trajectory, current\_state)}$
   $history\_states = calculated\_states;$
  $history\_control = calculated\_control;$
  $current\_state = next\_inputs;$
end

Algorithm 5: Inner optimization

$\boldsymbol{function}(seed\_trajectory, current\_state);$
Define stage master(template stage):
$assign\_vehicle\_model\_and\_extra\_variables;$
$assign\_path\_constraints : limitations\_vehicle;$
$assign\_path\_objective\_and\_solving\_method;$
Create stage: $stage = stage\_master$
 $stage\_start\_condition = current\_state;$
if $line\_segment$ **then**
  $set\_guess(x, y);$
  $set\_stay\_on\_lane\_constraint;$
  $set\_end\_stage\_requirement$
else
  $set\_guess(x, y);$
  $set\_stay\_on\_lane\_constraint;$
  $set\_end\_stage\_requirement$
end
$Stage\_previous = stage;$
for *i in [0:1:len(total_trajectory)]* **do**
  $end\_constraints\_stage\_previous = start\_constraints\_stage;$
  if $line\_segment$ **then**
    $set\_guess(x, y);$
    $set\_stay\_on\_lane\_constraint;$
    $set\_end\_stage\_requirement$
  else
    $set\_guess(x, y);$
    $set\_stay\_on\_lane\_constraint;$
    $set\_end\_stage\_requirement$
  end
  $Stage\_previous = stage;$
end
$Final\_constraints;$
$sol = ocp.solve()$

Algorithm 6: Path generation

Result: $\boldsymbol{r}_{opti}$
Initialization:
$\boldsymbol{\theta} = [1, 1, ..., 1, 1]$ Objective: