**FINAL GROUP PROJECT 2022**

# INVERTED PENDULUM CONTROL

# PERANCANGAN SISTEM KENDALI MODERN

Logo

Description automatically generated

**KELOMPOK 2**

**Disusun Oleh:**

Andreas Ryan Cahyo Kartiko (20/456085/TK/50215)

Bonaventura Riko K. D. (20/XXXXXX/TK/XXXXX)

Muhammad Bagus H. (20/XXXXXX/TK/XXXXX)

Rafli (20/XXXXXX/TK/XXXXX)

Yosef Adi Sulistyo (20/XXXXXX/TK/XXXXX)

**DEPARTEMEN TEKNIK ELEKTRO DAN TEKNOLOGI INFORMASI**

**FAKULTAS TEKNIK UNIVERSITAS GADJAH MADA**

**2022**

1. ***State Equation* dari *Inverted Pendulum***

Pada *group project* ini, akan dicoba untuk mengendalikan inverted pendulum sesuai *challenge* yang dideskripsikan. Adapun model dari inverted pendulum yang digunakan adalah sebagai berikut:

Diagram

Description automatically generated

Gambar 1. Model sistem *cart-pole*

* State variable dari model inverted pendulum di atas adalah:
* Model matematis dari referensi yang disediakan adalah sebagai berikut:

Dikarenakan model yang digunakan tidak linear, maka harus dilakukan linearisasi sebagai berikut:

Maka didapatkan model linear sebagai berikut:

Kemudian definisi dari masing-masing *state variable* nya menjadi:

Maka *derivative* nya menjadi:

* Didapatkan *state equation* nya menjadi:

Sesuai dengan *initial value* yang diberikan yaitu sebagai berikut:

Didapatkan *state equation* nya menjadi:

1. ***Challenge 1: Full-State Compensator***

|  |
| --- |
| import gym  import numpy as np  import control  l = 0.5  mp = 0.1  mc = 1.0  g = 9.8  dt = 0.02 # from openai gym docs  # get environment  env = gym.make('CartPole-v1', render\_mode="human").unwrapped  #env.env.seed(1) # seed for reproducibility  obs, info = env.reset(seed=1)  reward\_threshold = 475  reward\_total = 0  # System State Space Equation  A = np.array([[0, 1, 0, 0],  [0, 0, -mp\*(mp \* (g-l) + mc\*g)/((mc+mp)\*((4/3) \* mc + (1/3) \* mp)), 0],  [0, 0, 0, 1],  [0, 0, (mp\*(g-l) + mc \* g)/(l\*((4/3) \* mc + (1/3) \* mp)), 0]])  B = np.array([[0],  [(1/(mc + mp) - mp/((mc + mp) \* ((4/3) \* mc + (1/3) \* mp)))],  [0],  [(-1/(l \* ((4/3) \* mc + (1/3) \* mp)))]])  C = np.array([[1, 0, 0, 0],  [0, 0, 1, 0]])  At = np.transpose(A)  Bt = np.transpose(C)  Ct = np.transpose(B)  # desired pole  P = np.array([-10, -0.25+0.25j, -0.25-0.25j, -20])  Pt = 4\*P  # compute regulator and observer gain  K = control.place(A, B, P)  L = control.place(At, Bt, Pt)  L = np.transpose(L)  def compute\_state\_estimator(x\_hat, x, u):  y = C@x  x\_hat\_dot = A@x\_hat + B@u + L@(y - C@x\_hat)  x\_hat = x\_hat\_dot \* dt + x\_hat  return x\_hat  def apply\_state\_controller(x):  u = -K@x # u = -Kx  if u > 0:  action = 1  else:  action = 0  return action, u  obs\_hat = np.zeros(4)  print(obs\_hat)  u\_array = []  theta\_array = []  t\_array = []  for i in range(1000):  # time logging  t = i\*dt  t\_array.append(t)  env.render()  # states data logging  print("obs: ", obs)  print("obs\_hat: ", obs\_hat)  theta\_array.append(obs[2])  # MODIFY THIS PART  action, force = apply\_state\_controller(obs\_hat)  print("u: ", force)  u\_array.append(force)  # absolute value, since 'action' determines the sign, F\_min = -10N, F\_max = 10N  clip\_force = np.clip(force, -10, 10)  abs\_force = np.abs(float(clip\_force))  # change magnitute of the applied force in CartPole  env.force\_mag = abs\_force  # apply action  obs, reward, done, truncated, info = env.step(action)  # compute state estimator  obs\_hat = compute\_state\_estimator(obs\_hat, obs, clip\_force)  print()  reward\_total = reward\_total+reward  if done or truncated or reward\_total == reward\_threshold:  print(f'Terminated after {i+1} iterations.')  print("reward: ", reward\_total)  u\_array\_abs = []  for i in range(len(u\_array)):  u\_array\_abs.append(np.abs(u\_array[i]))  u\_avg = np.around(np.mean(u\_array\_abs),3)  print("force\_avg: ", u\_avg, "N")  theta\_min = np.amin(theta\_array)  theta\_max = np.amax(theta\_array)  if theta\_max > np.abs(theta\_min):  theta\_abs = theta\_max  search\_theta = theta\_max  else:  theta\_abs = np.abs(theta\_min)  search\_theta = theta\_min  overshoot\_rad = np.around(theta\_abs, 3)  overshoot\_deg = np.around(np.rad2deg(theta\_abs),3)  print("overshoot: ", overshoot\_deg, "degree")  for i in range(len(theta\_array)):  if np.abs(theta\_array[i]) < 1e-3:  peak\_time = np.around(i \* dt,3)/2  print("peak\_time: ", peak\_time, "s")  break  obs, info = env.reset()  break  env.close() |

1. ***Challenge 2: Reduced-Order Compensator***

|  |
| --- |
| """  References: https://pages.jh.edu/piglesi1/Courses/454/Notes6.pdf  Implemented in python with numpy  """  import gym  import numpy as np  import control  l = 0.5  mp = 0.1  mc = 1.0  g = 9.8  dt = 0.02 # from openai gym docs  # get environment  env = gym.make('CartPole-v1', render\_mode="human").unwrapped  #env.env.seed(1) # seed for reproducibility  obs, info = env.reset(seed=1)  reward\_threshold = 475  reward\_total = 0  # System State Space Equation  A = np.array([[0, 1, 0, 0],  [0, 0, -mp\*(mp \* (g-l) + mc\*g)/((mc+mp)\*((4/3) \* mc + (1/3) \* mp)), 0],  [0, 0, 0, 1],  [0, 0, (mp\*(g-l) + mc \* g)/(l\*((4/3) \* mc + (1/3) \* mp)), 0]])  B = np.array([[0],  [(1/(mc + mp) - mp/((mc + mp) \* ((4/3) \* mc + (1/3) \* mp)))],  [0],  [(-1/(l \* ((4/3) \* mc + (1/3) \* mp)))]])  C = np.array([[1, 0, 0, 0],  [0, 0, 1, 0]])  # reorganize matrices  Ar = np.array(A, copy=True)  Ar[[0, 1, 2, 3]] = Ar[[0, 2, 1, 3]]  Ar[:, [1, 2]] = Ar[:, [2, 1]]  Br = np.array(B, copy=True)  Br[[0, 1, 2, 3]] = Br[[0, 2, 1, 3]]  Cr = np.array(C, copy=True)  Cr[:,[1,2]] = Cr[:,[2,1]]  # partitioned matrices  # a = available states  # u = unavailable states  Aaa = Ar[:2,:2]  Aau = Ar[:2,2:]  Aua = Ar[2:,:2]  Auu = Ar[2:,2:]  Ba = Br[:2]  Bu = Br[2:]  # desired poles  P = np.array([-0.25+0.25j, -0.25-0.25j, -20, -30])  Pt = 6 \* P[:2]  # compute regulator and observer gains  K = control.place(A, B, P)  L = control.place(np.transpose(Auu), np.transpose(Aau), Pt)  L = np.transpose(L)  def compute\_reduced\_observer(x, x\_hat, y, xcc, u):  xcopy = np.array(x, copy=True)  xa = np.empty([2,])  xa[[0]] = xcopy[[0]]  xa[[1]] = xcopy[[2]]  print("xa: ", xa)  x\_hat\_copy = np.array(x\_hat, copy=True)  xu\_hat = np.empty([2,])  xu\_hat[[0]] = x\_hat\_copy[[1]]  xu\_hat[[1]] = x\_hat\_copy[[3]]  print("xu\_hat: ", xu\_hat)  xcc\_dot = (Auu - L@Aau)@xu\_hat + (Aua - L@Aaa)@y + (Bu - L@Ba)@u  xcc = xcc + xcc\_dot\*dt  xu\_hat = xcc + L@y    x\_hat\_new = np.concatenate((xa, xu\_hat))  x\_hat\_new[[2,1]] = x\_hat\_new[[1,2]]  return xcc,x\_hat\_new    def apply\_state\_controller(x):  u = -K@x # u = -Kx  if u > 0:  action = 1  else:  action = 0  return action, u  obs\_hat = np.zeros(4,)  xcc = np.zeros(2,)  u\_array = []  theta\_array = []  t\_array = []  for i in range(1000):  # time logging  t = i\*dt  t\_array.append(t)  env.render()  # states data logging  print("obs\_hat: ", obs\_hat)  print("obs: ", obs)  theta\_array.append(obs[2])  # MODIFY THIS PART  action, force = apply\_state\_controller(obs\_hat)  print("u:", force)  u\_array.append(force)  # absolute value, since 'action' determines the sign, F\_min = -10N, F\_max = 10N  clip\_force = np.clip(force, -10, 10)  abs\_force = np.abs(float(clip\_force))  # change magnitute of the applied force in CartPole  env.force\_mag = abs\_force  # apply action  obs, reward, done, truncated, info = env.step(action)  y = C@obs  # compute observer state  xcc,obs\_hat = compute\_reduced\_observer(obs, obs\_hat, y, xcc, clip\_force)  print("obs\_hat: ", obs\_hat)  print()  reward\_total = reward\_total+reward  if done or truncated or reward\_total == reward\_threshold:  print(f'Terminated after {i+1} iterations.')  print("reward: ", reward\_total)  u\_array\_abs = []  for i in range(len(u\_array)):  u\_array\_abs.append(np.abs(u\_array[i]))    u\_avg = np.around(np.mean(u\_array\_abs),3)  print("force\_avg: ", u\_avg, "N")  theta\_max = np.amax(theta\_array)  theta\_min = np.amin(theta\_array)  if theta\_max > np.abs(theta\_min):  theta\_abs = theta\_max  search\_theta = theta\_max  else:  theta\_abs = np.abs(theta\_min)  search\_theta = theta\_min    overshoot\_rad = np.around(theta\_abs, 3)  overshoot\_deg = np.around(np.rad2deg(theta\_abs),3)  print("overshoot: ", overshoot\_deg, "degree")  for i in range(len(theta\_array)):  if np.abs(theta\_array[i]) < 1e-3:  peak\_time = np.around(i \* dt,3)/2  print("peak\_time: ", peak\_time, "s")  break  obs, info = env.reset()  break  env.close() |

1. ***Challenge 3: Robust Tracking***

|  |
| --- |
| import gym  import numpy as np  import control  l = 0.5  mp = 0.1  mc = 1.0  g = 9.8  dt = 0.02  # get environment  env = gym.make('CartPole-v1', render\_mode="human").unwrapped  #env.env.seed(1) # seed for reproducibility  obs, info = env.reset(seed=1)  reward\_threshold = 475  reward\_total = 0  # System State Space Equation  A = np.array([[0, 1, 0, 0],  [0, 0, -mp\*(mp \* (g-l) + mc\*g)/((mc+mp)\*((4/3) \* mc + (1/3) \* mp)), 0],  [0, 0, 0, 1],  [0, 0, (mp\*(g-l) + mc \* g)/(l\*((4/3) \* mc + (1/3) \* mp)), 0]])  B = np.array([[0],  [(1/(mc + mp) - mp/((mc + mp) \* ((4/3) \* mc + (1/3) \* mp)))],  [0],  [(-1/(l \* ((4/3) \* mc + (1/3) \* mp)))]])  C = np.array([[1, 0, 0, 0]])  # Augmented SS Equation for Robust Tracking  A\_aug = np.block([[np.zeros([C.shape[0],C.shape[0]]), C],  [np.zeros([A.shape[0],C.shape[0]]), A]])  print("A\_aug: ", A\_aug)  B\_aug = np.block([[np.zeros([C.shape[0],1])],  [B]])  print("B\_aug: ", B\_aug)  B\_L = np.array(B\_aug, copy=True)  # noise/disturbance  w = np.array([0.5])  w = np.reshape(w,1)  # desired pole  P = np.array([-0.25+0.5j, -0.25-0.5j, -10, -20])  P\_aug = np.array([-2+0.5j,-2-0.5j,-1.75+0.25j,-1.75-0.25j,-100])  # compute regulator gain  K = control.place(A,B,P)  K\_aug = control.place(A\_aug, B\_aug, P\_aug)  print("K\_aug: ", K\_aug)  def f\_aug\_linear(x, u):  x\_aug\_dot = A\_aug@x + B\_aug@u + B\_L@u  return x\_aug\_dot  def apply\_state\_controller(x):  # feedback controller  # MODIFY THIS PARTS  if(x.shape[0] == A\_aug.shape[0]):  K\_cont = K\_aug  else:  K\_cont = K    u = -K\_cont @ x  print("u: ", u)  return u  obs\_aug = np.block([[np.zeros([C.shape[0],1])],  [obs.reshape([4,1])]])  obs\_aug = np.reshape(obs\_aug, obs\_aug.shape[0])  force = np.zeros([1,])  u\_array = []  x\_array = []  theta\_array = []  t\_array = []  for i in range(1000):  # time logging  t = i\*dt  t\_array.append(t)  env.render()    # log state  x\_array.append(obs[0])  theta\_array.append(obs[2])  print("obs: ", obs)  print("obs\_aug: ", obs\_aug)  # MODIFY THIS PART  force = apply\_state\_controller(obs\_aug)  # input noise  force = force + w    # log force  u\_array.append(force)  # determine action  if force > 0:  action = 1  else:  action = 0  # absolute value, since 'action' determines the sign, F\_min = -10N, F\_max = 10N  abs\_force = abs(float(np.clip(force, -10, 10)))    # change magnitute of the applied force in CartPole  env.force\_mag = abs\_force  # apply action  obs, reward, done, truncated, info = env.step(action)  obs\_aug\_dot = f\_aug\_linear(obs\_aug, force)  obs\_aug = obs\_aug + obs\_aug\_dot \* dt  for n in range(obs.shape[0]):  obs\_aug[n+C.shape[0]] = obs[n]  reward\_total = reward\_total+reward  print()  if done or truncated or reward\_total == reward\_threshold:  print(f'Terminated after {i+1} iterations.')  print("reward: ", reward\_total)  x\_array\_abs = []  for i in range(len(x\_array)):  x\_array\_abs.append(abs(x\_array[i]))    ess\_avg = np.around(np.mean(x\_array\_abs),3)  print("ess: ", ess\_avg)  u\_array\_abs = []  for i in range(len(u\_array)):  u\_array\_abs.append(abs(u\_array[i]))  u\_avg = np.around(np.mean(u\_array\_abs),3)  print("u\_avg: ", u\_avg, "N")  theta\_min = np.amin(theta\_array)  theta\_max = np.amax(theta\_array)  if theta\_max > np.abs(theta\_min):  theta\_abs = theta\_max  search\_theta = theta\_max  else:  theta\_abs = np.abs(theta\_min)  search\_theta = theta\_min    overshoot\_rad = np.around(theta\_abs, 3)  overshoot\_deg = np.around(np.rad2deg(theta\_abs),3)  print("overshoot: ", overshoot\_deg, "degree")  for i in range(len(theta\_array)):  if np.abs(theta\_array[i]) < 1e-3:  peak\_time = np.around(i \* dt,3)/2  print("peak\_time: ", peak\_time, "s")  break  obs, info = env.reset()  break  env.close() |

1. ***Challenge 4: Optimal Control***

|  |
| --- |
| import gym  import numpy as np  from scipy import linalg  l = 0.5  mp = 0.1  mc = 1.0  g = 9.8  dt = 0.02  # get environment  env = gym.make('CartPole-v1', render\_mode="human").unwrapped  #env.env.seed(1) # seed for reproducibility  obs, info = env.reset(seed=1)  reward\_threshold = 475  reward\_total = 0  # ADD SOMETHING HERE  A = np.array([[0, 1, 0, 0],  [0, 0, -mp\*(mp \* (g-l) + mc\*g)/((mc+mp)\*((4/3) \* mc + (1/3) \* mp)), 0],  [0, 0, 0, 1],  [0, 0, (mp\*(g-l) + mc \* g)/(l\*((4/3) \* mc + (1/3) \* mp)), 0]])  B = np.array([[0],  [(1/(mc + mp) - mp/((mc + mp) \* ((4/3) \* mc + (1/3) \* mp)))],  [0],  [(-1/(l \* ((4/3) \* mc + (1/3) \* mp)))]])  C = np.array([[1, 0, 0, 0],  [0, 0, 1, 0]])  Q = np.array([[1, 0, 0, 0],  [0, 10, 0, 0],  [0, 0, 1, 0],  [0, 0, 0, 100]])  R = np.array([[0.1]])  P = linalg.solve\_continuous\_are(A, B, Q, R)  K = np.linalg.inv(R)@np.transpose(B)@P  def apply\_state\_controller(K, x):  # feedback controller  # MODIFY THIS PARTS  u = -np.dot(K, x) # u = -Kx  print("u: ", u)  if u > 0:  action = 1  else:  action = 0  return action, u  u\_array = []  theta\_array = []  t\_array = []  for i in range(1000):  # time logging  t = i\*dt  t\_array.append(t)  env.render()  # print current state  print("obs: ", obs)  theta\_array.append(obs[2])  # get force direction (action) and force value (force)  # MODIFY THIS PART  action, force = apply\_state\_controller(K, obs)  u\_array.append(force)  # absolute value, since 'action' determines the sign, F\_min = -10N, F\_max = 10N  abs\_force = abs(float(np.clip(force, -10, 10)))    # change magnitute of the applied force in CartPole  env.force\_mag = abs\_force  # apply action  obs, reward, done, truncated, info = env.step(action)  reward\_total = reward\_total+reward  print()  if done or truncated or reward\_total == reward\_threshold:  print(f'Terminated after {i+1} iterations.')  print("reward: ", reward\_total)  u\_array\_abs = []  for i in range(len(u\_array)):  u\_array\_abs.append(np.abs(u\_array[i]))  u\_avg = np.around(np.mean(u\_array\_abs),3)  print("u\_avg: ", u\_avg, "N")  theta\_min = np.amin(theta\_array)  theta\_max = np.amax(theta\_array)  if theta\_max > np.abs(theta\_min):  theta\_abs = theta\_max  search\_theta = theta\_max  else:  theta\_abs = np.abs(theta\_min)  search\_theta = theta\_min    overshoot\_rad = np.around(theta\_abs, 3)  overshoot\_deg = np.around(np.rad2deg(theta\_abs),3)  print("overshoot: ", overshoot\_deg, "degree")  for i in range(len(theta\_array)):  if np.abs(theta\_array[i]) < 1e-3:  peak\_time = np.around(i \* dt,3)/2  print("peak\_time: ", peak\_time, "s")  break  obs, info = env.reset()  break  env.close() |

Link Source Code di GitHub:

<https://github.com/yosefadi/Final_Project_PSKM>