Sprawozdanie do projektu 1
Alexander Krasovskiy
Aleksander Kaluta

Wstęp

W niniejszym sprawozdaniu opisane zostało rozwiązanie projektu 1, które dotyczyło zaimplementowania modelu poruszającego się ramienia. Rozwiązanie projektu opisywane w tym sprawozdaniu spełnia wszystkie wymagania podstawowe i wymagania dodatkowe a), połowe b) oraz c).

Organizacja projektu

Kod źródłowy zawierający implementację modelu poruszającego się ramienia wraz graficznym interfejsem użytkownika dostępny jest w udostępnionym pliku Lab2.zip. W folderze graphics zostały umieszczone grafiki używane w implementacji zadania.

Rozwiązanie

Ramię składa się z trzech segmentów i węzłów, a na końcu ostatniego segmentu znajduje się manipulator M w formie chwytaka.

```
21          self.vertices = (M, w3, w2, w1)
```

Pierwszy węzeł jest zaczepiony stale w punkcie w1 (0,0)

```
172     manipulator = Manipulator((0, 0), (0, 20), (0, 40), (0, 70))
```

Program demonstruje ciągły ruch w zakresie możliwym do manipulacji w kontrolce tekstowej. Startowo każdy węzeł porusza się po kątach w zakresie [-180, 180] stopni.

```
50      Entry(self.window, bg="#FFFFFF", fg="#000000", textvariable=self.lim3min,
51      Entry(self.window, bg="#FFFFFF", fg="#000000", textvariable=self.lim3max,
```

```
117         try:
118             angle_lims = [
119                 (self.lim1min.get(), self.lim1max.get()),
120                 (self.lim2min.get(), self.lim2max.get()),
121                 (self.lim3min.get(), self.lim3max.get()),
122                 (self.lim4min.get(), self.lim4max.get()),
123             ]
124             for lim in angle_lims:
125                 assert lim[0] <= 0 and lim[1] >= 0
126
127             manipulator.angle_limits = angle_lims
```

```
26          if params is None:
27              params = [
28                  tuple([(-180, 180)] + [(-180, 180) for _ in range(len(self._rotate) - 1)]),
29              ]
30          else:
31              assert len(params) == 1
32              assert len(params[0]) == len(self._rotate)
33
34          self.angle_limits = params[0]
```

Każdy segment ma ustaloną długość. którą można zmieniać poprzez suwak dostępny w okienku tk. Dla przykładu suwak s1 oddaje zmienną zmiennoprzecinkową self.scale0

```
36          s1 = Scale(self.window, from_=1, to=self.scale_max,
37                     resolution=0.01, variable=self.scale0)
```

```
13          self.scale0 = DoubleVar()
```

Następnie tą zmienną używa w funkcji scale

```
101         try:
102             if self.scale0.get() != 1:
103                 manipulator.scale(0, self.scale0.get())
104         except TclError:
105             pass
```

Funkcja scale zmienia element listy _scale o indeksie zero na zmieniony przez suwak współczynnik.

```
24          self._scale = [1, 1, 1]
```

```
107     def scale(self, vert_num, coeff):
108         self._scale[vert_num] = coeff
```

Kolejnie funkcja _eval_scaled_points oddaje przeskalowany wektor V.

```
117
118     def _eval_scaled_points(self, V):
119         for _i in range(1, len(V)):
120             vec_x = V[_i][0] - V[_i - 1][0]
121             vec_y = V[_i][1] - V[_i - 1][1]
122             vec_x = vec_x * self._scale[_i - 1] - vec_x
123             vec_y = vec_y * self._scale[_i - 1] - vec_y
124             for j in range(_i, len(V)):
125                 V[j][0] += vec_x
126                 V[j][1] += vec_y
127         return V
```

Analogicznie prędkość rotacji segmentów można zmienić poprzez kontrolkę spinbox. Dla przykładu pierwszy spinbox oddaje zmienną całkowitą self.rotate0.

```
33          Spinbox(self.window, textvariable=self.rotate0,
```

```
16          self.rotate0 = IntVar()
```

Następnie tą zmienną używa w funkcji rotate

```
80  try:
81      if self.rotate0.get() != 0 and abs(self.rotate0.get()) <= 90:
82          manipulator.rotate(0, self.rotate0.get())
83  except TclError:
84      pass
```

Funkcja scale działa jedynie na element o indeksie 0 listy _rotate, gdy kontrolka checkbutton jest włączona, a w innych sytuacjach zmienia element listy _rotate o wybranym indeksie na zmieniony przez spinbox kąt. Gdy kąt jest równy 180 zmienia wartość na -180, przez co zachowuje swoją orientację.

```
84  def rotate(self, vert_num, angle):
85      if self.lock and vert_num > 0:
86          return
87      self._rotate[vert_num] += angle
88      min_r, max_r = self.angle_limits[vert_num]
89      self._rotate[vert_num] = min(max_r, max(min_r, self._rotate[vert_num]))
90      if abs(self._rotate[vert_num]) >= 180:
91          self._rotate[vert_num] = -self._rotate[vert_num]
```

Funkcja rotate_around_point obraca punkt względem danego punktu.

```
53      @staticmethod
54      def rotate_around_point(xy, radians, origin=(0, 0)):
55          """Rotate a point around a given point.
56          """
57          x, y = xy
58          offset_x, offset_y = origin
59          adjusted_x = (x - offset_x)
60          adjusted_y = (y - offset_y)
61          cos_rad = cos(radians)
62          sin_rad = sin(radians)
63          qx = offset_x + cos_rad * adjusted_x + sin_rad * adjusted_y
64          qy = offset_y + -sin_rad * adjusted_x + cos_rad * adjusted_y
65          return [qx, qy]
```

Funkcja evalrotated_points obraca wektor o dany kąt z listy _rotate.

```
110     def _eval_rotated_points(self):
111         V = [x for x in self.vertices]
112         for k in range(len(self._rotate)):
113             r = self._rotate[k]
114             for _i in range(k + 1, len(V)):
115                 V[_i] = Manipulator.rotate_around_point(V[_i], r / 180 * math.pi, V[k])
116         return V
```

Funkcja _eval_vertices skaluje obrócony wektor.

Funkcja loadAssets wczytuje grafiki

```
41     def loadAssets(self):
42         self.hand = pygame.image.load(Manipulator.graphics_path + Manipulator._hand)
43         self.arm = pygame.image.load(Manipulator.graphics_path + Manipulator._arm)
```

```
10     _hand = "/hand.png"
11     _arm = "/arm.png"
12     graphics_path = "./graphics/"
```

Funkcja rotate_surface obraca daną powierzchnię wokół punktu.

```
93     @staticmethod
94     def rotate_surface(surface, angle, pivot, offset):
95         """Rotate the surface around the pivot point.
96         Args:
97             surface (pygame.Surface): The surface that is to be rotated.
98             angle (float): Rotate by this angle.
99             pivot (tuple, list, pygame.math.Vector2): The pivot point.
100            offset (pygame.math.Vector2): This vector is added to the pivot.
101        """
102        rotated_image = pygame.transform.rotozoom(surface, -angle, 1)
103        rotated_offset = offset.rotate(angle)
104        rect = rotated_image.get_rect(center=pivot + rotated_offset)
105        return rotated_image, rect
```

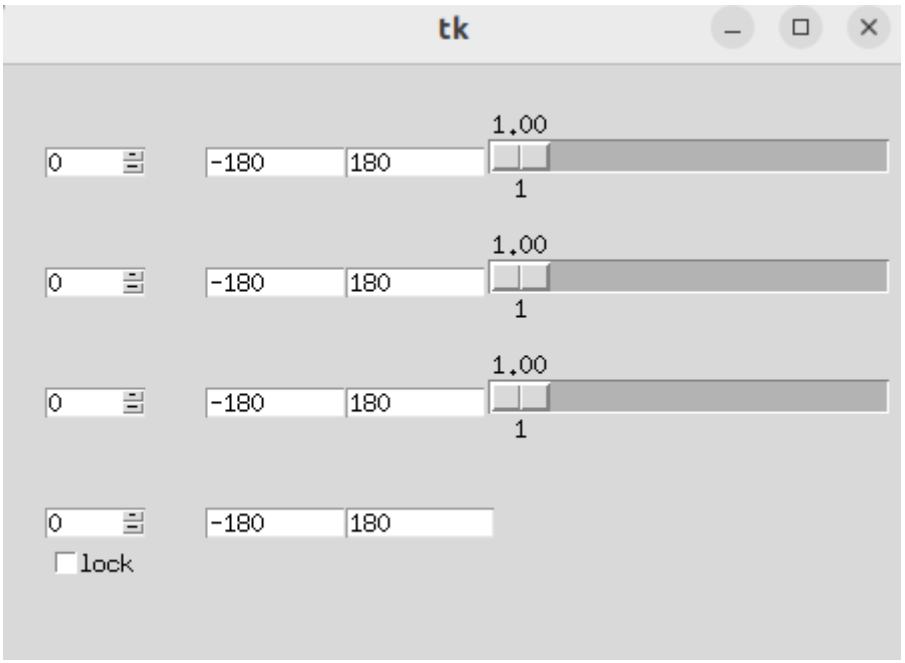
Funkcje _draw_arm oraz _draw_hand definiują wartości potrzebne do rysowania segmentów.

```
129    @staticmethod
130    def _draw_arm(surface: pygame.Surface, V1, V2, arm_width, arm=None, ):
131        if arm is None:
132            pygame.draw.line(surface, "black", V1, V2)
133        else:
134            angle = Manipulator.angle(V1, V2)
135            dist = Manipulator.distance(V1, V2)
136            arm = pygame.transform.scale(arm, (arm_width, dist * 1.1))
137            surface.blit(*Manipulator.rotate_surface(arm, angle * 180 / math.pi, V1, pygame.math.Vector2(0, -dist / 2)))
138
139    def _draw_hand(self, surface: pygame.Surface, V, V1, hand_width, hand=None):
140        if hand is None:
141            pygame.draw.line(surface, "blue", V, V1)
142        else:
143            self._draw_arm(surface, V, V1, hand_width, hand)
```

Funkcja draw rysuje po kolei wszystkie segmenty.

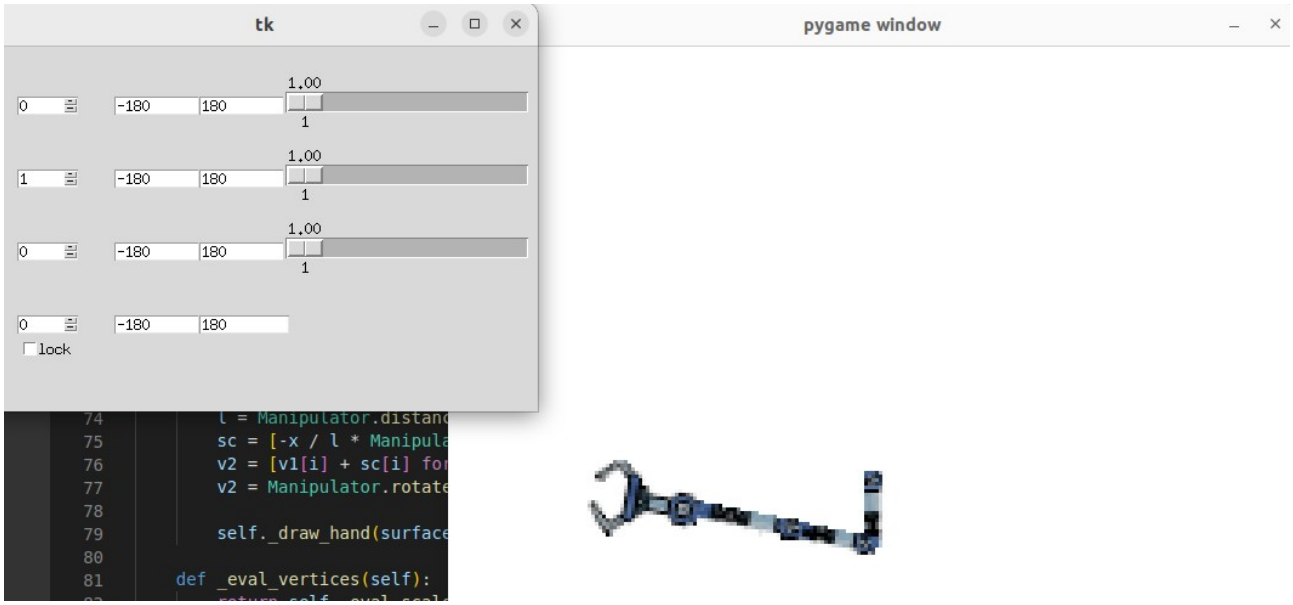
```python
    def draw(self, surface: pygame.Surface):
        vertices = self._eval_vertices()
        for _i in range(1, len(vertices)):
            Manipulator._draw_arm(surface, vertices[_i - 1], vertices[_i], Manipulator.arm_width, self.hand)

        v1 = vertices[3]
        sc = [vertices[2][0] - v1[0], vertices[2][1] - v1[1]]
        l = Manipulator.distance((0, 0), sc)
        sc = [-x / l * Manipulator.hand_width * Manipulator.hand_ratio for x in sc]
        v2 = [v1[i] + sc[i] for i in range(len(v1))]
        v2 = Manipulator.rotate_around_point(v2, self._rotate[3] / 180 * math.pi, v1)

        self._draw_hand(surface, v1, v2, Manipulator.hand_width, self.arm)
```

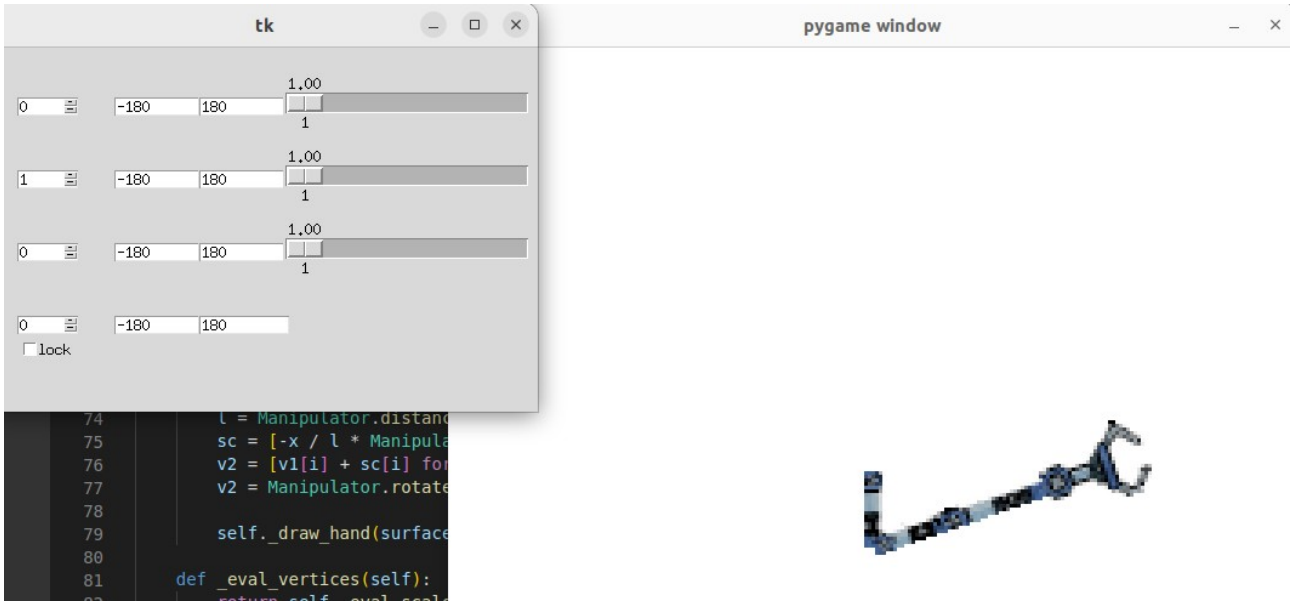
Klasa ManipulatorGUI tworzy kontrolki ułatwiające korzystanie z programu.



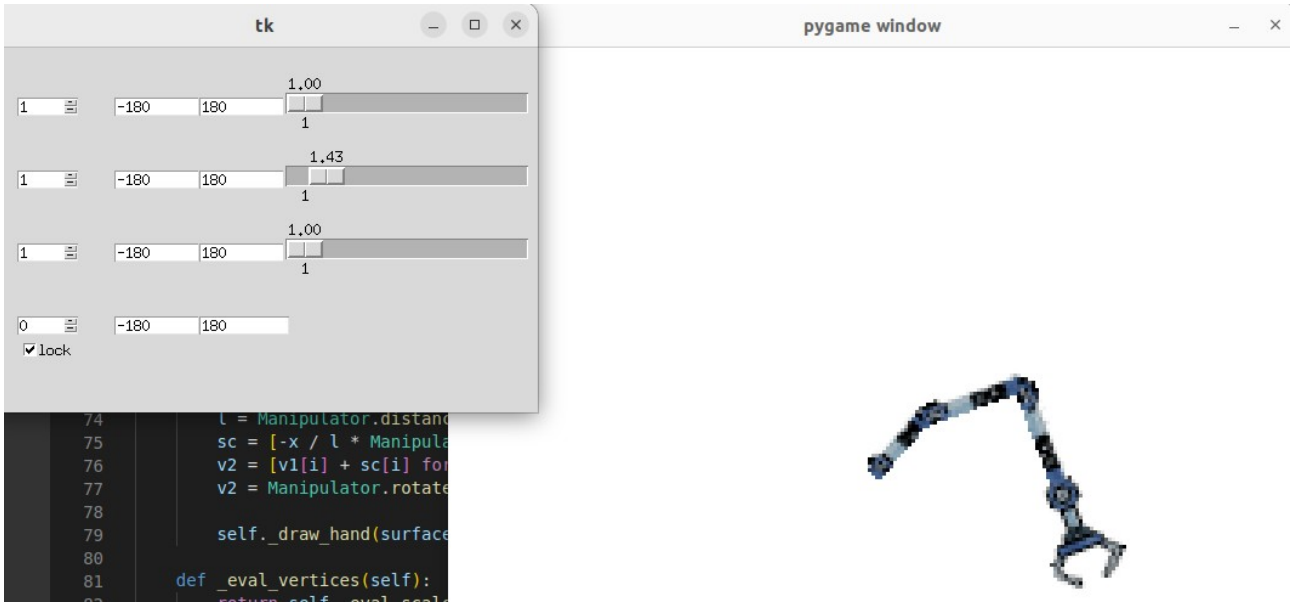
Przykładowe rozwiązania układu:

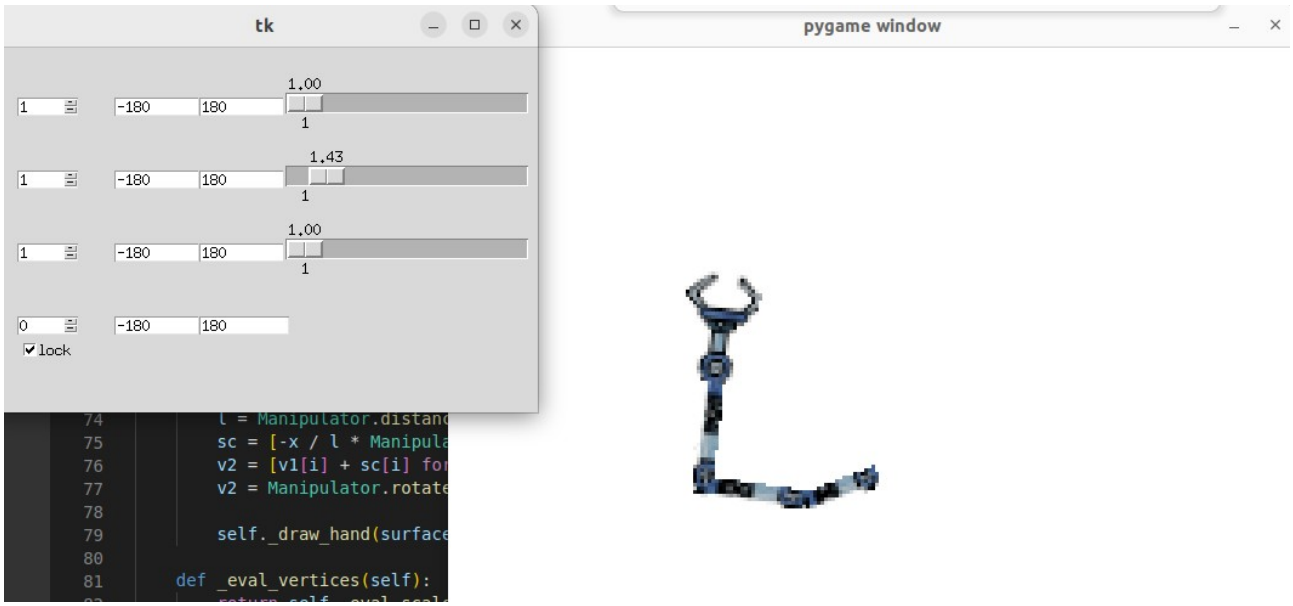
1.




tk
pygame window
1.00
1.00
1.00
lock
l = Manipulator.distanc
sc = [-x / l * Manipula
v2 = [v1[i] + sc[i] for
v2 = Manipulator.rotate
self._draw_hand(surface
def _eval_vertices(self):

2.


tk
pygame window
1.00
1.43
1.00
lock
l = Manipulator.distanc
sc = [-x / l * Manipula
v2 = [v1[i] + sc[i] for
v2 = Manipulator.rotate
self._draw_hand(surface
def _eval_vertices(self):


tk
pygame window
1.00
1.43
1.00
lock
l = Manipulator.distanc
sc = [-x / l * Manipula
v2 = [v1[i] + sc[i] for
v2 = Manipulator.rotate
self._draw_hand(surface
def _eval_vertices(self):