Require: getObjCoords, px2xy, motorSync, invKin, magOn, magOff

1: $boxCoords \leftarrow [-91, 120; 91, 120; -91, 45; 91, 45]$
2: $centroids \leftarrow$ getObjCoords()
3: $[numObj, \_\,] \leftarrow$ size($centroids$)
4: $[\varphi_{L0}, \varphi_{R0}] \leftarrow$ invKin$(0, 29.3)$
5: $\varphi_L \leftarrow \varphi_{L0}$
6: $\varphi_R \leftarrow \varphi_{R0}$
7: $k \leftarrow 1$
8: **for** $i \leftarrow 1$ to $numObj$ **step** 1 **do**
9: $[x, y] \leftarrow$ px2xy$(centroids(i, 1), centroids(i, 2))$
10: $[\varphi_{Lr}, \varphi_{Rr}] \leftarrow$ invKin(x, y)
11: $\Delta\varphi_L \leftarrow -(\varphi_{Lr} - \varphi_L)$
12: $\Delta\varphi_R \leftarrow -(\varphi_{Rr} - \varphi_R)$
13: motorSync$(\Delta\varphi_L, \Delta\varphi_R)$
14: $\varphi_L \leftarrow \varphi_{Lr}$
15: $\varphi_R \leftarrow \varphi_{Rr}$
16: magOn()
17: $[\varphi_{Lr}, \varphi_{Rr}] \leftarrow$ invKin$(boxCoords(k, 1), boxCoords(k, 2))$
18: $\Delta\varphi_L \leftarrow -(\varphi_{Lr} - \varphi_L)$
19: $\Delta\varphi_R \leftarrow -(\varphi_{Rr} - \varphi_R)$
20: motorSync$(\Delta\varphi_L, \Delta\varphi_R)$
21: $\varphi_L \leftarrow \varphi_{Lr}$
22: $\varphi_R \leftarrow \varphi_{Rr}$
23: magOff()
24: $k \leftarrow k + 1$
25: **if** mod$(i, 4) = 0$ **then**
26: $k \leftarrow 1$
27: **end if**
28: **end for**
29: $[\varphi_{Lr}, \varphi_{Rr}] \leftarrow$ invKin$(0, 29.3)$
30: $\Delta\varphi_L \leftarrow -(\varphi_{Lr} - \varphi_L)$
31: $\Delta\varphi_R \leftarrow -(\varphi_{Rr} - \varphi_R)$
32: motorSync$(\Delta\varphi_L, \Delta\varphi_R)$

Fig. A.1 Pseudocode for general pick and place task

Require: φ_L, φ_R, setStepL, setStepR, impSeq

```
1:  procedure MOTORSYNC(φ_L, φ_R)
2:      ratio ← 64    st_rev ← 64    stepCount ← 8    stepTime ← 2/1000
3:      seqL ← impSeq(φ_L)    seqR ← impSeq(φ_R)
4:      revsL ← |φ_L|/(2π)    stepsL ← round(revsL · st_rev · ratio)
5:      revsR ← |φ_R|/(2π)    stepsR ← round(revsR · st_rev · ratio)
6:      if stepsL ≥ stepsR then
7:          dStep ← round(stepsL/stepsR)
8:          j ← 1    jj ← 1
9:          for i ← 1 to stepsR step 1 do
10:             setStepR(seqR(j,1), seqR(j,2), seqR(j,3), seqR(j,4))
11:             for ii ← 1 to dStep step 1 do
12:                 setStepL(seqL(jj,1), seqL(jj,2), seqL(jj,3), seqL(jj,4))
13:                 delay(stepTime)
14:                 if jj = stepCount then
15:                     jj ← 1
16:                 else
17:                     jj ← jj + 1
18:                 end if
19:             end for
20:             if j = stepCount then
21:                 j ← 1
22:             else
23:                 j ← j + 1
24:             end if
25:         end for
26:     else
27:         dStep ← round(stepsR/stepsL)
28:         j ← 1    jj ← 1
29:         for i ← 1 to stepsL step 1 do
30:             setStepL(seqL(j,1), seqL(j,2), seqL(j,3), seqL(j,4))
31:             for ii ← 1 to dStep step 1 do
32:                 setStepR(seqR(jj,1), seqR(jj,2), seqR(jj,3), seqR(jj,4))
33:                 delay(stepTime)
34:                 if jj = stepCount then
35:                     jj ← 1
36:                 else
37:                     jj ← jj + 1
38:                 end if
39:             end for
40:             if j = stepCount then
41:                 j ← 1
42:             else
43:                 j ← j + 1
44:             end if
45:         end for                    3
46:     end if
47: end procedure
```

Fig. A.2 Pseudocode for synchronous motor movement – motorSync function

Require: φ

```
1: procedure impSeq(φ)
2:     if φ > 0 then
3:         seq(1,:) ← [1,0,0,0]
4:         seq(2,:) ← [1,1,0,0]
5:         seq(3,:) ← [0,1,0,0]
6:         seq(4,:) ← [0,1,1,0]
7:         seq(5,:) ← [0,0,1,0]
8:         seq(6,:) ← [0,0,1,1]
9:         seq(7,:) ← [0,0,0,1]
10:        seq(8,:) ← [1,0,0,1]
11:    else if φ < 0 then
12:        seq(8,:) ← [1,0,0,0]
13:        seq(7,:) ← [1,1,0,0]
14:        seq(6,:) ← [0,1,0,0]
15:        seq(5,:) ← [0,1,1,0]
16:        seq(4,:) ← [0,0,1,0]
17:        seq(3,:) ← [0,0,1,1]
18:        seq(2,:) ← [0,0,0,1]
19:        seq(1,:) ← [1,0,0,1]
20:    else
21:        seq ← zeros(8,4)
22:    end if
        return seq
23: end procedure
```

Fig. A.3 Pseudocode for stepper motor half step operation sequence – impSeq function

Require: A, B, A_p, B_p

```
1: procedure setStepL(A, B, Ap, Bp)
2:     writeDigitalPin(14, Bp)        ▷ blue / Bp
3:     writeDigitalPin(15, A)         ▷ pink / A
4:     writeDigitalPin(23, B)         ▷ yellow / B
5:     writeDigitalPin(24, Ap)        ▷ orange / Ap
6: end procedure
7: procedure setStepL(A, B, Ap, Bp)
8:     writeDigitalPin(25, Bp)        ▷ blue / Bp
9:     writeDigitalPin(12, A)         ▷ pink / A
10:    writeDigitalPin(16, B)         ▷ yellow / B
11:    writeDigitalPin(20, Ap)        ▷ orange / Ap
12: end procedure
```

Fig. A.4 Pseudocode for functions for setting the stepper motors pins state

procedure GETOBJCOORDS()
  rgbImg $\leftarrow$ snapshot($'/dev/video0'$,$'$ 640x480$'$)
  gsImg $\leftarrow$ rgb2gray($rgbImg$)
  $[i_m, j_m] \leftarrow$ size($gsImg$)
  for $i \leftarrow 1$ to i_m **step** 1 **do**
    for $j \leftarrow 1$ to j_m **step** 1 **do**
      if $j \geq 98$ and $j \leq 585$ and $i \geq 11$ and $i \leq 412$ **then**
        $\mathbf{M}(i, j) \leftarrow 1$
      else
        $\mathbf{M}(i, j) \leftarrow 0$
      end if
    end for
  end for
  $\mathbf{K} \leftarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
  $\mathbf{H} \leftarrow$ conv2(rgbImage(:, :, 1), $\mathbf{K}$,$'$same$'$)
  $\mathbf{V} \leftarrow$ conv2(rgbImage(:, :, 1), $\mathbf{K}^{-1}$,$'$same$'$)
  $\mathbf{E} \leftarrow \sqrt{\mathbf{H} \odot \mathbf{H} + \mathbf{V} \odot \mathbf{V}}$
  $\mathbf{B_w} \leftarrow (\mathbf{E} > 25) \odot \mathbf{M}$
  $\mathbf{B_o} \leftarrow$ bwareaopen($\mathbf{B_w}$, 3750)
  $\mathbf{B_f} \leftarrow$ imfill($\mathbf{B_o}$,$'$holes$'$)
  $hBlob \leftarrow$ vision.BlobAnalysis()
  $centroid \leftarrow hBlob(\mathbf{B_f})$
  return $centroids$
end procedure

Fig. A.5 Pseudocode for obtaining the object coordinates

Require: x_{im}, y_{im}

procedure PX2XY(x_{im}, y_{im})
  $scale \leftarrow 0.2410$
  $v_x \leftarrow 340 \cdot scale$
  $v_y \leftarrow 212 \cdot scale$
  $t_y \leftarrow 95$
  $x \leftarrow x_{im} \cdot scale - v_x$
  $y \leftarrow -y_{im} \cdot scale + v_y + t_y$
  return x, y
end procedure

Fig. A.6 Pseudocode for transforming pixels to milimeters – px2xy function

Require: x_{TCP}, y_{TCP}

procedure INVKIN(x_{TCP}, y_{TCP})
  $l_1 \leftarrow 75$
  $l_2 \leftarrow 95$
  $e \leftarrow 22.5$
  $\varphi_R \leftarrow \text{atan2}(y_{TCP}, x_{TCP} - e) - \text{acos}\left(\frac{(x_{TCP}-e)^2+y_{TCP}^2+l_1^2-l_2^2}{2l_1\sqrt{(x_{TCP}-e)^2+y_{TCP}^2}}\right)$
  $\varphi_L \leftarrow \text{atan2}(y_{TCP}, x_{TCP} + e) - \text{acos}\left(\frac{(x_{TCP}+e)^2+y_{TCP}^2+l_1^2-l_2^2}{2l_1\sqrt{(x_{TCP}+e)^2+y_{TCP}^2}}\right)$
  return φ_R, φ_L
end procedure

Fig. A.7 Pseudocode for inverse kinematics – invKin function

Require: maskThresh

```
1: procedure DOTDETECT(())
2:     rgbImg ← snapshot('/dev/video0','640x480')
3:     R ← rgbImg(:,:,1)
4:     G ← rgbImg(:,:,2)
5:     B ← rgbImg(:,:,3)
6:     redThresholdLow ← 230
7:     redThresholdHigh ← 255
8:     greenThresholdLow ← 84
9:     greenThresholdHigh ← 125
10:    blueThresholdLow ← 130
11:    blueThresholdHigh ← 170
12:    M_R ← maskThresh(R, redThresholdLow, redThresholdHigh)
13:    M_G ← maskThresh(G, greenThresholdLow, greenThresholdHigh)
14:    M_B ← maskThresh(B, blueThresholdLow, blueThresholdHigh)
15:    M_P ← M_R ⊙ M_G ⊙ M_B
16:    hBlob ← vision.BlobAnalysis()
17:    centroid ← hBlob(M_P)
18:    return centroids
19: end procedure
```

Fig. A.8 Pseudocode for tool centger point color detection

Require: $band, threshLo, threshHi$

```
1: procedure MASKTHRESH(band, threshLo, threshHi)
2:     [i_m, j_m] ← size(band)
3:     for i ← 1 to i_m step 1 do
4:         for j ← 1 to j_m step 1 do
5:             if band(i,j) ≤ threshHi and band(i,j) ≥ threshLo then
6:                 colorMask(i,j) ← 1
7:             else
8:                 colorMask(i,j) ← 0
9:             end if
10:        end for
11:    end for
12:    return colorMask
13: end procedure
```

Fig. A.9 Pseudocode for defining color mask