Instituto Tecnológico y de Estudios Superiores de Monterrey

TE3002B.502

Implementación de robótica inteligente (Gpo 101)

Semestre: febrero - junio 2023

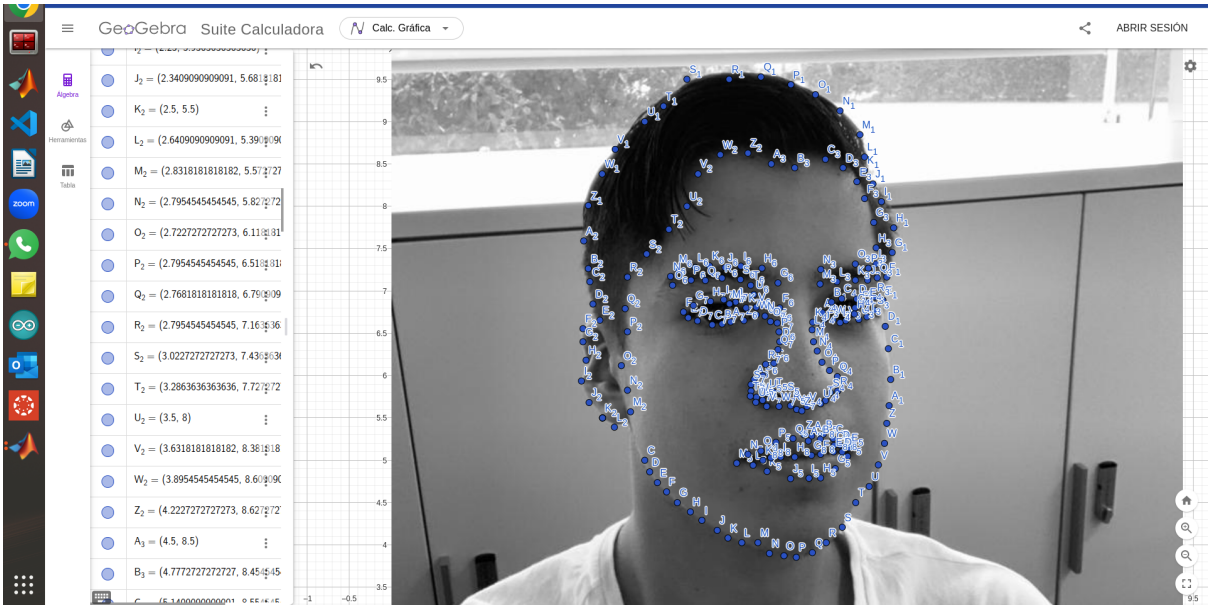
Evaluación 7.2

Alumno:

Daniel Ruán Aguilar A01731921

Profesor: Dr. Alfredo García Suárez

En esta evaluación se obtiene el mapa de puntos para generar la planeación de trayectorias a partir de mi rostro, marcando set-points con el software Geogebra a partir de la siguiente foto:



Posteriormente se planea la trayectoria que debe seguir el algoritmo empleando el mapa de puntos obtenidos, considerando la descripción más próxima a los rasgos físicos de mi rostro.

Se implementó el código requerido para generar la planeación de trayectorias, la técnica que se consideró más eficiente para esta aplicación fue **PurePursuit (way-points)**, ya que hicimos un ejercicio previamente de formar figuras a partir de coordenadas dadas. Lo que hace el algoritmo es que a partir de un vector de pares ordenados, sigue la trayectoria en el orden que se fueron declarando los puntos, por lo que para esta actividad se consideró muy útil.

Se realizó el siguiente ajuste de los parámetros cinemáticos del robot móvil:

Velocidad angular (MaxAngularVelocity = 23): Se aumentó en gran cantidad la velocidad angular, ya que de esta manera la trayectoria llega completamente a cada punto y los giros son más precisos y cerrados, lo cual da un mejor resultado en este caso que la trayectoria contiene muchas curvas.

velocidad lineal (DesiredLinearVelocity = 0.3): Por otro lado se disminuyó en gran medida la velocidad lineal para lograr tramos más curvos entre cada punto.

Un ancho de la muestra (sampleTime = 0.1): El tamaño de las muestras se mantuvo con el mismo valor de 0.1 segundos, es decir con 10 muestras entre cada segundo, para evitar “saltos” y que el movimiento fuera más fluido.

Tiempo de simulación (tVec = 116.5): Iterando manualmente, el tiempo final que le tomaría hacer toda la trayectoria del rostro resultó ser de 116.5 segundos con los parámetros previamente establecidos.

Posición inicial (initPose = [3;5;pi/2]): Para la posición de inicio, se concluyó que poniendo la primera coordenada con un giro inicial de 90° sería la mejor manera de empezar y pasar al siguiente waypoint.

Distancia de anticipación (controller.LookaheadDistance = 0.15): Finalmente, como en este ejercicio se incluyeron muchos waypoints con una muy pequeña distancia entre cada uno, se disminuyó la distancia de anticipación para que no se omitiera ninguno en la trayectoria, si los waypoints hubieran estado con más

separación entre ellos, se hubiera aumentado este parámetro para que fuera capaz de visualizar el siguiente punto.

Para saber el número de waypoints declarados, se usó la función length():

```
>> length(waypoints)

ans =

   132
```

El Código en MATLAB con los 132 waypoints queda de la siguiente manera:

```
%% Define Vehicle
R = 0.1;                % Wheel radius [m]
L = 0.5;                % Wheelbase [m]
dd = DifferentialDrive(R,L);

%% Simulation parameters
sampleTime = 0.1;               % Sample time [s]
tVec = 0:sampleTime:116.5;         % Time array
initPose = [3;5;pi/2];             % Initial pose (x y theta)
pose = zeros(3,numel(tVec));    % Pose matrix
pose(:,1) = initPose;

% Define waypoints
waypoints = [3,5; 3.15,4.73636; 3.38636,4.5; 3.67727,4.290909; 3.98636,
4.0818; 4.3409, 4.027272; 4.65,3.872727; 4.986, 3.90909; 5.340909, 4.20909;
5.65909,4.6909; 5.8409,5.2; 5.89545, 5.64545; 5.913636,5.9545; 5.886,6.31818;
5.8227,6.8909; 5.931818, 7.45454; 5.95,7.74545; 5.8045,8.0545; 5.6045,
8.42727;5.604545,8.581818;
   5.55, 8.84545; 5.3136,9.12727; 5.0227, 9.31818; 4.73181,9.43636; 4.37727,
9.527272;   4,   9.5;   3.5,   9.5;3.2227272727273,   9.181818;3,   9;2.65,
8.6727;2.4954545454545,    8.3818;2.3318181818182,    8.009;2.2772727272727,
7.590;2.35,          7.109;2.4681818181818,          6.654;2.2681818181818,
6.55;2.2681,6.4;2.3045454545455,   6.1818;2.25,   5.936;2.340909,5.6818;2.5,
5.5;2.6409, 5.390;
     2.8318, 5.572727; 2.795454,5.82727; 2.722, 6.118; 2.79545, 6.51818;
2.76818,6.7909;    2.79545,7.1636;    3.022727,7.4363;    3.2863,7.72727;3.5,
8;3.6318,8.3818;3.89545,                          8.6090;4.2227,8.62727;4.5,
```

```
8.5;4.7772727,8.4545;5.1409,                                    8.554545;5.35,
8.45;5.6045,8.0909;5.71363,7.80909;5.740,          7.509;5.7409,7.318;5.6409,
7.127;5.27727, 7.109;5.077,7.0818;
                                      5.08636,7.2545;           5.49545,7.318;
5.64949,7.26868;5.7545,7.15;5.726,6.77;5.633,6.6949;           5.3828,6.6505;
5.2333,6.6585;      5.104,      6.7434;      5.213,6.8686;5.3343,6.90909;5.512,
6.9;5.49,6.8;5.483,6.727;5.314,    6.626;4.99,6.63;5,    6.4;5.1,    6.15;5.188,
6.06;5.281,5.98;5.28,5.818;5.2,    5.8;5.0838,5.66;4.92,5.636;4.88,5.30;4.978,
5.28; 5.116, 5.309;5.314, 5.17;

5.419,5.147;5.403,5.046;5.0838,4.79;4.728,4.783;4.2838,4.941;4.089,4.96;4.223
,5.07;4.554, 5.212; 4.75, 5.256;4.72,5.64; 4.663, 5.74; 4.445,  5.789; 4.396,
5.7090;4.259,          5.818;4.295,5.955;4.4,          6;4.6,          6.4;4.6,
6.8;4.5787,7.0949;4.388,7.268;4.138,7.2929;3.766,7.2969;3.59,7.276;3.30,7.17;
3.54,7.127;3.9161,7.151; 4.134,7.135; 4.29,6.95
                  4.32,6.8;      4.324,6.719;4.174,6.65;3.916,      6.61;3.8,
6.6;3.625,6.64;3.459,6.75;3.576,6.82;3.93,6.89;4.0131,6.84;4.0050, 6.74];

% Create visualizer
viz = Visualizer2D;
viz.hasWaypoints = true;

%% Pure Pursuit Controller
controller = controllerPurePursuit;
controller.Waypoints = waypoints;
controller.LookaheadDistance = 0.15;
controller.DesiredLinearVelocity = 0.3;
controller.MaxAngularVelocity = 23;

%% Simulation loop,
close all
r = rateControl(1/sampleTime);
for idx = 2:numel(tVec)
   % Run the Pure Pursuit controller and convert output to wheel speeds
   [vRef,wRef] = controller(pose(:,idx-1));
   [wL,wR] = inverseKinematics(dd,vRef,wRef);

   % Compute the velocities
   [v,w] = forwardKinematics(dd,wL,wR);
   velB = [v;0;w]; % Body velocities [vx;vy;w]
   vel = bodyToWorld(velB,pose(:,idx-1));  % Convert from body to world
```

```
    % Perform forward discrete integration step
    pose(:,idx) = pose(:,idx-1) + vel*sampleTime;

    % Update visualization
    viz(pose(:,idx),waypoints)
    waitfor(r);
    grid on
    hold on
end
```

Resultado final del seguimiento de trayectoria de mi rostro