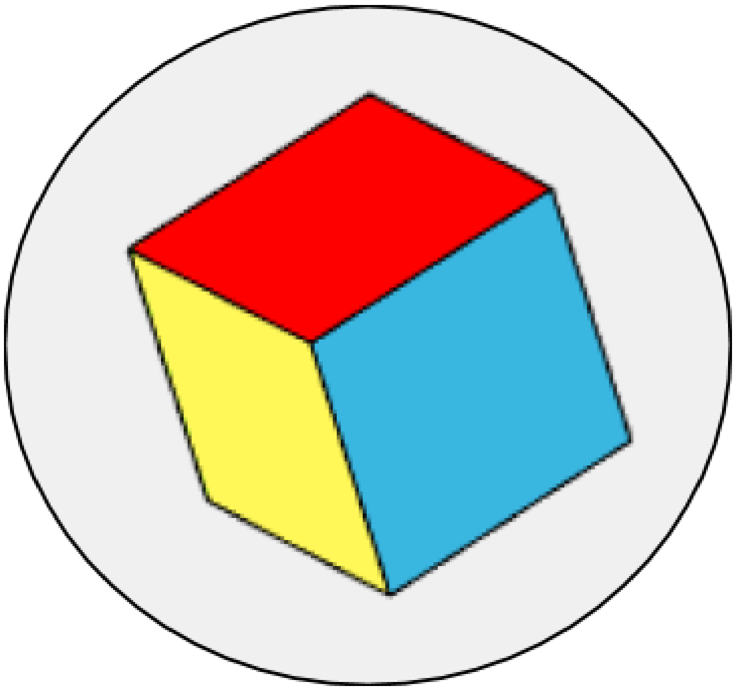
Matlab Final Project
Trackball



Pablo Galve Millán
Marc Pagès Francesch
Carlos Redolar Torres
Maths II
CITM
5th of January of 2020

Introduction

The purpose of this project is to create a trackball using the GUIDE tool form Matlab. Taking as an input the movement of the mouse it will provide as outputs a graphical representation and five numerical parametrizations of the attitude of the cube.

List of functions

These are the functions that we have used for the calculations of the attitude of the cube and the transformation to other attitude parametrizations.

calculateM

This function's purpose is to transform the 2D coordinates given by the user using the mouse to 3D coordinates in the imaginary sphere surrounding the cube.
In order to avoid abrupt rotations when calculating the surface of the sphere this function combines both the equations of the sphere and the hyperboloid and depending on the X and Y of the mouse it takes one equation or the other to calculate the Z coordinate.

quaternionFromVectors

If we create a quaternion using the sine and cosine of the angle formed by two vectors rotating around an axis and develop a little bit the mathematical equivalences we can reach this function's implementation. It simplifies the calculations avoiding the sine and cosine and just doing a square root, two norms, one cross product and a dot product.
The two vectors we take as inputs aren't vectors actually, they are the points on the screen the user is clicking projected onto the sphere and the previous point clicked. As outputs it will give the quaternion representing the necessary rotation to go from one point to the other and which can be used to get the next quaternion that represents the actual orientation of the cube.

quaternionMultiplication

This function takes as inputs two quaternions and outputs their product. It is used to calculate the current orientation of the cube due to the fact that if we take the delta quaternion transformed to a unit quaternion and multiply it to the previous orientation we get the actual orientation of the cube.

transformAttitudes

Once we know the current attitude of the cube as a quaternion we can use the following functions to transform it to other parametrizations and set the input boxes strings to their corresponding values. This function combines some of them.

quaternion2AxisAngle

This function takes as an input a quaternion and transforms it to its corresponding attitude equivalence in euler's axis and angle. It gets the angle by multiplying for two the inverse cosine of it's scalar part and dividing it's vectorial part by half of the sine. In the case in which the angle is zero it outputs the x axis but it's value could have been any other else because the axis has no relevance if there is no angle to rotate around.

rotm2eAngles

A rotation matrix is taken as input and the three equivalent Euler Angles are given as an output. It takes in account the special cases in which the angle is 0, 90 and 270 degrees in which case if the normal equations were used and it wouldn't output a number.

quaternion2RotationMatrix

Using equivalences at the time of rotating a vector using a unit quaternion this matrix can be deducted and is the one this function uses to transform a unit quaternion into a rotation matrix which will be used to output the attitude equivalence or to redraw the cube.

rotateMatVec

This function takes as input a rotation vector and outputs its equivalent rotation matrix. First, we normalize the vector and if the result is 0, we then return the identity matrix. Otherwise we use the Rodrigues' rotation formula to get the rotation matrix.

Eaa2rotMat

If we take as inputs an angle which represents the rotation around an axis and use the Rodrigues rotation formula we can create a matrix which represents the same rotation.

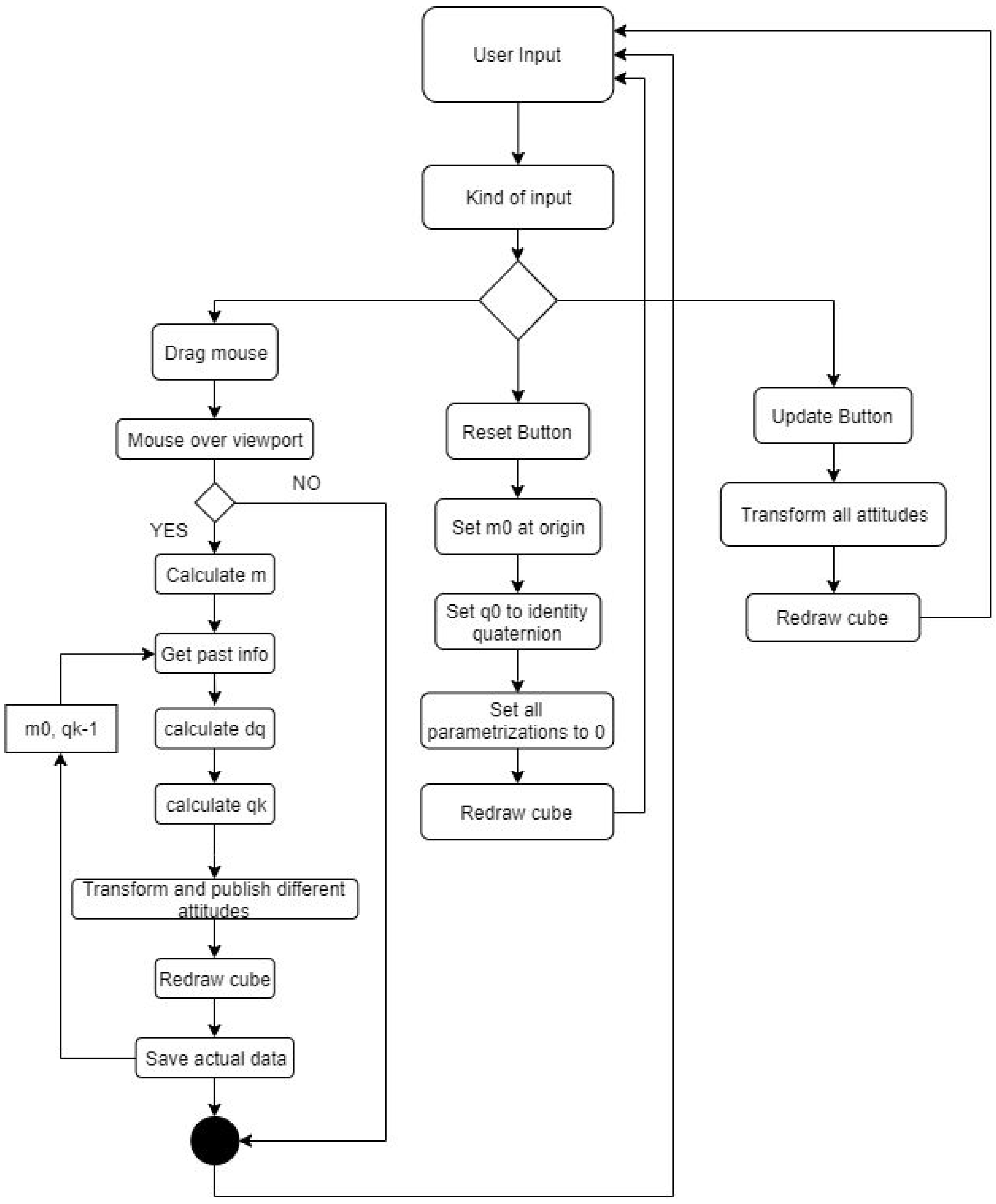
eAngles2rotM

If we combined the rotation matrices that rotate a vector an angle around the X axis, the angle rotated around the resulting Y axis and the angle rotated around the resulting Z axis the result would be the matrix we use in this function to create the rotation matrix used to rotate a vector using three Euler Angles.
The sine and cosine of the three angles are computed before putting them into the rotation matrix to avoid unnecessary calculations given the fact that they are static values.

rotMat2Eaa

This function calculates the angle that the Matrix rotates the vector around the axis and the axis. To get the angle we take the general formula which implies the trace of the matrix and to calculate the axis we first check if the angle is 0 or 180 to avoid future problems. If the angle is 0 it outputs a random axis given the fact that to represent the rotation of 0 the axis has no relevance. If the angle is 180 degrees we first calculate the ux and then the axis using this. In the cases in which the angle is different from 0 or 180 we just use the normal procedure to calculate the axis.

User diagram



These are the equivalences between the steps and the functions used:

- Calculate m: calculateM
- calculate dq: quaternionFromVectors
- Caclulate qk: quaternionMultiplication
- Transform and publish different attitudes: transformAttitudes

Transform all Attitudes

Depending on the update button different functions were used:

- Quaternions Update
  - transformAttitudes
  - quaternion2RotationMatrix
- Euler Angle and Axis Update
  - Eaa2rotMat
  - rotM2eAngles
- Euler Angles Update
  - eAngles2rotM
  - rotMat2Eaa
- Rotation Vector Update
  - rotateMatVec
  - rotM2eAngles

You can check the video explaining the full functioning of the Arcball here:
https://youtu.be/HhrUOnl_vw8