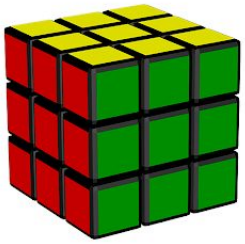


Gálvez Llorens, Marc
Martínez de la Rosa, Christian
Moreu Farran, Lluís
Peña Hernando , Carlos

MATHS II- CITM

ARCBALL

Visualizing 3D data

MATLAB FINAL PROJECT

Index

Implemented functions

[M] = Eaa2rotMat(u,angle)

This function uses the euler axis and angle to make a rotation matrix thanks to the Rodrigues' rotation formula:

$$R(u, \varphi) = I_3 * \cos \varphi + (1 - \cos(\varphi)) (u * u^T) + [u]_x * \sin(\varphi)$$

INPUTS:

- Angle in radians "angle"
- Column axis "u"

OUTPUT:

- Rotation matrix "M"

[ang,unitary] = rotMat2Eaa(R)

This function does the opposite than the one described above, that given a rotation matrix returns its respective euler principal column axis and angle in radians. It's possible due to the inverse mapping of the Rodrigues' rotation formula because, given the symmetries and anti-symmetries of the terms in the formula we can calculate the angle and the axis.

INPUT:

- Rotation matrix "R"

OUTPUTS:

- Angle in degrees "ang"
- Column axis "unitary"

[rotM] = eAngles2rotM(phi, theta, psy)

This function uses a given a set of euler angles in degrees and returns its respective rotation matrix.
The function is the result of rotate the global frame about the z axis direction an angle ψ, then rotate the global frame about the y axis direction an angle θ and finally rotate the global frame about the x axis direction an angle φ, so is the multiplication of the 3 matrix.

INPUTS:

- Angle "phi"
- Angle "theta"
- Angle "psy"

OUTPUT:

- Rotation matrix "rotM"

[phi1,phi2,theta1,theta2,psi1,psi2,flag] = rotM2eAngles(R)

This function does the opposite than the one described above, given a rotation matrix, returns its respective euler angles. This is possible thanks to the inverse mapping of the formula given above, where the term located in the third row-first column helps to find the first angle, then using atan2 we locate the other angles. But in this project the secondary angles and the flag are ignored.

INPUT:

- Rotation matrix “R”

OUTPUTS:

- Angle “phi1 and Angle “phi2”
- Angle “theta1” and Angle “theta2”
- Angle “psi1” and Angle “psi2”
- Flag “flag”

[R] = V2rotMat(vec)

This function uses a Rotation Vector to get a rotation matrix using the Rodrigues’ Rotation Formula.

R(v) = I3 * cos (norm_v) + ((1 − cos(norm_v))/ norm_v2) (v* vT) + [u]x * (sin(norm_v) / norm_v)

INPUTS:
Rotation vector “vec”

OUTPUT:
Rotation matrix “R”

[q3] = multiplyQuat(q1,q2)

This function uses two quaternions and multiplies them, giving a result Quaternion.
In order to do this, it uses this formula:

$$q^{\circ}p^{\circ} = (q0 * p0 - qt * p / q0 * p + p0 * q + q \times p)$$

· q0 is called its scalar part
· q its vector part.
· qt as the transpose matrix of q

INPUTS:

- Quaternion “q1”
- Quaternion “q2”

OUTPUT:

- Result Quaternion “q3”

[R] = quat2RotMat(q)

This function transforms a quaternion to a rotation matrix.
In order to do this, it uses the rotation matrix composed by using quaternions:

We want to rotate a vector $\boldsymbol{v}$ into a new vector $\boldsymbol{w}$ using a unit quaternion $\mathring{q}$:

- Step 1: Insert the vector $\boldsymbol{v}$ in the shape of a pure quaternion, $\mathring{v} = (0\,, \boldsymbol{v}^\intercal)^\intercal$
- Step 2: Calculate $\mathring{v}' = \mathring{q}\mathring{v}\tilde{\mathring{q}} = \mathbf{Q}_L(\mathring{q})\mathbf{Q}_R(\tilde{\mathring{q}})\mathring{v}$ with

$$\mathbf{Q}_L(\mathring{q})\mathbf{Q}_R(\tilde{\mathring{q}}) = \begin{pmatrix} \star & \star_3^\intercal \\ \star_3 & \mathbf{R}(\mathring{q}) \end{pmatrix}$$

$$\mathbf{R}(\mathring{q}) = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix}$$

INPUT:

- Quaternion “q”

OUTPUT:

- Rotation matrix “R”

[module] = quat_module(quad)

This is a function where the module of a quaternion is calculated . It is the same for calculating the module of a vector, calculating the square root of the sum of all of its terms squared.

INPUT:

- Quaternion “quad”

OUTPUT:

- Module of the quaternion “module”

[quat_normalized] = quat_normalize(q)

This is a function where a quaternion is normalized . It is the same for normalizing a vector., dividing the quaternion for its module.

INPUT:

- Quaternion “q”

OUTPUT:
- Module of the quaternion “quat_normalized”

[v] = Eaa2V(angle,axis)

This is a function where the Euler axis and angle are transformed into a rotation vector. The procedure is simple, you only need to multiply the axis by the angle.

INPUTS:
- Euler Angle “angle”
- Euler Axis “axis

OUTPUT:
- Rotation vector “v”

[q] = rotMat2quat(R)

This is a function where a rotation matrix is transformed into a quaternion. Operating with the diagonal terms of the rotation matrix expressed in quat2RotMat (L) and applying the unity norm constraint, the following can be obtained:

$$
\begin{aligned}
4q_0^2 &= 1 + l_{11} + l_{22} + l_{33} = 1 + \operatorname{trace}(\mathbf{L}) \\
4q_1^2 &= 1 + l_{11} - l_{22} - l_{33} = 1 - \operatorname{trace}(\mathbf{L}) + 2l_{11} \\
4q_2^2 &= 1 - l_{11} + l_{22} - l_{33} = 1 - \operatorname{trace}(\mathbf{L}) + 2l_{22} \\
4q_3^2 &= 1 - l_{11} - l_{22} + l_{33} = 1 - \operatorname{trace}(\mathbf{L}) + 2l_{33}
\end{aligned}
$$

Then making some assumptions we get to these 4 cases:

Case1 : In case that $q_0^2 > q_i^2$ for $i = 1, 2, 3$, which is achieved iff trace $(\mathbf{L}) \geq 0$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \sqrt{1 + l_{11} + l_{22} + l_{33}} \\ \frac{l_{32} - l_{23}}{\sqrt{1+l_{11}+l_{22}+l_{33}}} \\ \frac{l_{13} - l_{31}}{\sqrt{1+l_{11}+l_{22}+l_{33}}} \\ \frac{l_{21} - l_{12}}{\sqrt{1+l_{11}+l_{22}+l_{33}}} \end{bmatrix} \tag{8}$$

Case2 : In case that $q_1^2 > q_i^2$ for $i = 0, 2, 3$ which is achieved iff trace $(\mathbf{L}) < 0$ and $l_{11} = \max(l_{11}, l_{22}, l_{33})$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \frac{l_{32} - l_{23}}{\sqrt{1+l_{11}-l_{22}-l_{33}}} \\ \sqrt{1 + l_{11} - l_{22} - l_{33}} \\ \frac{l_{21} + l_{12}}{\sqrt{1+l_{11}-l_{22}-l_{33}}} \\ \frac{l_{13} + l_{31}}{\sqrt{1+l_{11}-l_{22}-l_{33}}} \end{bmatrix} \tag{9}$$

Case3 : In case that $q_2^2 > q_i^2$ for $i = 0, 1, 3$ which is achieved iff trace $(\mathbf{L}) < 0$ and $l_{22} = \max(l_{11}, l_{22}, l_{33})$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \frac{l_{13} - l_{31}}{\sqrt{1-l_{11}+l_{22}+l_{33}}} \\ \frac{l_{21} + l_{12}}{\sqrt{1-l_{11}+l_{22}+l_{33}}} \\ \sqrt{1 - l_{11} + l_{22} + l_{33}} \\ \frac{l_{32} + l_{23}}{\sqrt{1-l_{11}+l_{22}+l_{33}}} \end{bmatrix} \tag{10}$$

Case4 : In case that $q_3^2 > q_i^2$ for $i = 0, 1, 2$ which is achieved iff trace $(\mathbf{L}) < 0$ and $l_{33} = \max(l_{11}, l_{22}, l_{33})$ based on Eq. (3)

$$\mathring{q} = \frac{1}{2} \begin{bmatrix} \frac{l_{21} - l_{12}}{\sqrt{1-l_{11}-l_{22}+l_{33}}} \\ \frac{l_{13} + l_{31}}{\sqrt{1-l_{11}-l_{22}+l_{33}}} \\ \frac{l_{32} + l_{23}}{\sqrt{1-l_{11}-l_{22}+l_{33}}} \\ \sqrt{1 - l_{11} - l_{22} + l_{33}} \end{bmatrix} \tag{11}$$

INPUT:

- Rotation matrix “R”

OUTPUT:

- Quaternion “q”

my_MouseClickFcn(obj,event,hObject)

This function receives as handle object the axes element where the cube is located. Once, we know the axes limits, according to the mouse position it will detect if the user makes a click on the cube. If that’s it, the last mouse position will be saved as a quaternion which its Z parameter is calculated by Holroyd’s arcball formulations encoded:

$$\boldsymbol{m}(x,y) = \begin{cases} \left(x,\, y,\, +\sqrt{r^2 - x^2 - y^2}\right)^{\intercal} & x^2 + y^2 < \frac{1}{2}r^2 \\ \dfrac{r\left(x,\, y,\, \frac{r^2}{2\sqrt{x^2+y^2}}\right)^{\intercal}}{\left\|\left(x,\, y,\, \frac{r^2}{2\sqrt{x^2+y^2}}\right)\right\|} & x^2 + y^2 \geq \frac{1}{2}r^2 \end{cases}$$

INPUTS:

- Figure “obj”
- Event “event”
- Handle to figure “hObject”

my_MouseReleaseFcn(obj,event,hObject)

This function receives as handle object the axes element which it has detected on “my_MouseClickFcn” method and setted off from handles structure of figure.

INPUTS:

- Figure “obj”
- Event “event”
- Handle to figure “hObject”

my_MouseMoveFcn(obj,event,hObject)

This function receives as handle object the axes element where the cube is located. Once, we know the axes limits, according to the mouse position it will detect of the user makes a click on the cube.

If that’s it, it has the same functionality that my_MouseClickFcn method with the addition of get the previous quaternion from last mouse position click and obtain the angle which is shared between both quaternions:

$$\theta = \arccos\left(\frac{\boldsymbol{m_1}^{\intercal}\boldsymbol{m_0}}{\|\boldsymbol{m_1}\|\,\|\boldsymbol{m_0}\|}\right)$$

After that, the quaternion that encodes the rotation can be calculated by the following formulation:

$$\mathring{q} = \left(\cos(\theta/2)\,,\, \sin(\theta/2)\,\frac{\boldsymbol{m_0}\times\boldsymbol{m_1}}{\|\boldsymbol{m_0}\times\boldsymbol{m_1}\|}\right)^{\intercal}$$

Therefore, we use our customized “quat2RotMat” which calculates the respective rotation matrix from previous quaternion and use it to display the new attitude of Cube by “RedrawCube” function.

handles.Cube = RedrawCube(R,handles.Cube);

INPUTS:

- Figure “obj”
- Event “event”
- Handle to figure “hObject”

h = DrawCube(R)

Function that receives a rotation matrix as input parameter to build the cube with his respective attitude.

INPUTS:

- Rotation matrix “R”

OUTPUT:

- h “h”

h = RedrawCube(R,hin)

Function that receives a rotation matrix and and the object to handle (axes) as input parameters to rebuild the new attitude of the cube.

INPUTS:

- Rotation matrix “R”
- hin “hin”

OUTPUT:

- h “h”

SetInitialVector(v)

Sets the initial_v as v.

INPUT:

- Vector “v”

v = GetInitialVector()

Sets v as the initial_v

OUTPUT:

- Vector “v”

SetInitialQuaternion(q)

Sets the prev_q as q.

INPUT:

- Quaternion “q”

q = GetInitialQuaternion()

Sets q as the prev_q.

OUTPUT:

- Quaternion “q”

SetRotationMatrix(R, handles)

Gets a Rotation matrix “R” and sets its valors in the editable texts. Also uses the appropriate functions to set the other values (Euler angle and axis, Euler angles, quaternion and rotation vector).

INPUTS:

- Rotation matrix “R”
- Handles “handles”

button_euler_angles_Callback(hObject, eventdata, handles)

Gets the Euler angles as a handle and use them to transform it as a rotation matrix, then calls SetRotationMatrix and RedrawCube.

INPUTS:

- handle to button_euler_angles_Callback “hObject”
- eventdata
- handles structure with handles and user data

button_axis_angle_Callback(hObject, eventdata, handles)

Gets the Euler angle and axis as a handle and use them to transform it as a rotation matrix, then calls SetRotationMatrix and RedrawCube.

INPUTS:

- handle to button_axis_angle_Callback“hObject”
- eventdata
- handles structure with handles and user data

button_quaternion_Callback(hObject, eventdata, handles)

Gets the quaternion terms as a handle and use them to transform it as a rotation matrix, then calls SetRotationMatrix and RedrawCube.

INPUTS:

- handle to button_quaternion_Callback“hObject”
- eventdata
- handles structure with handles and user data

button_rotation_vector_Callback(hObject, eventdata, handles)

Gets the rotation vector terms as a handle and use them to transform it as a rotation matrix, then calls SetRotationMatrix and RedrawCube.

INPUTS:

- handle to button_rotation_vector_Callback “hObject”
- eventdata
- handles structure with handles and user data

button_reset_Callback(hObject, eventdata, handles)

Calls the appropriate functions to set the initial vectors and quaternions as the beginning, then calls SetRotationMatrix and RedrawCube.

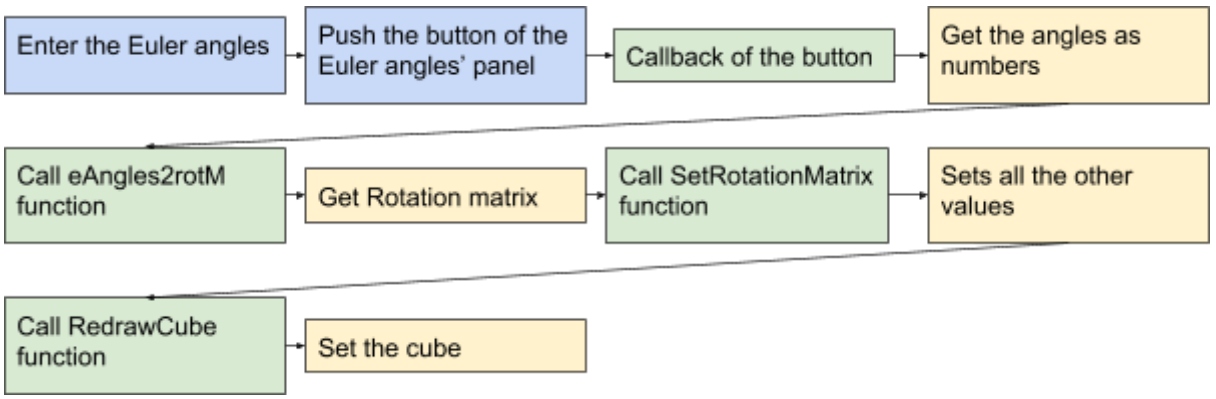
INPUTS:

- handle to button_reset_Callback“hObject”
- eventdata
- handles structure with handles and user data

User actions’ diagram

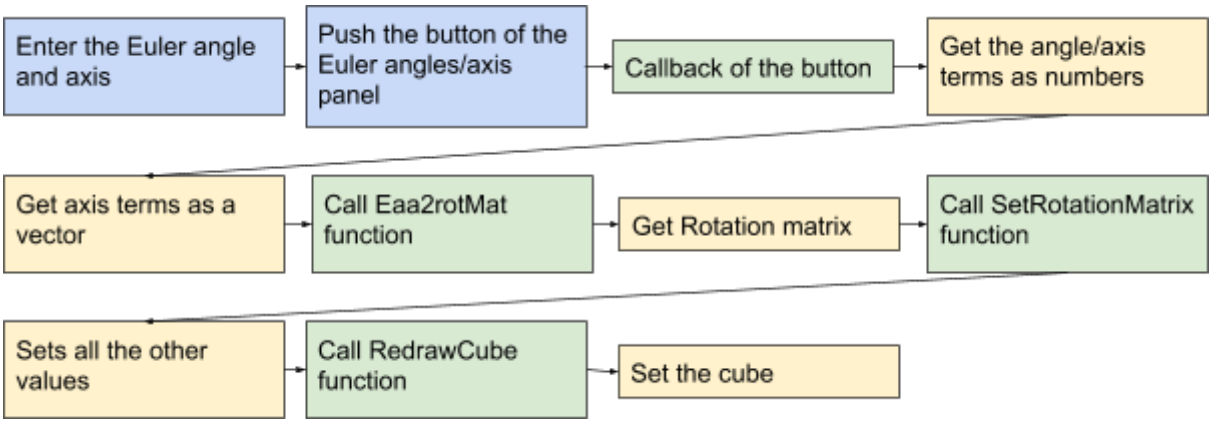
When the user enters in the figure, there are up to seven different options in the panel:

1. Euler Angles
    a. Enter three Euler Angles in the editable texts.
    b. Push the button of the Euler Angles’ panel.
    c. The callback of the button gets the angles as string and calls the function eAngles2rotM to get the rotation matrix, with this rotation matrix it calls RedrawCube to set the cube again.



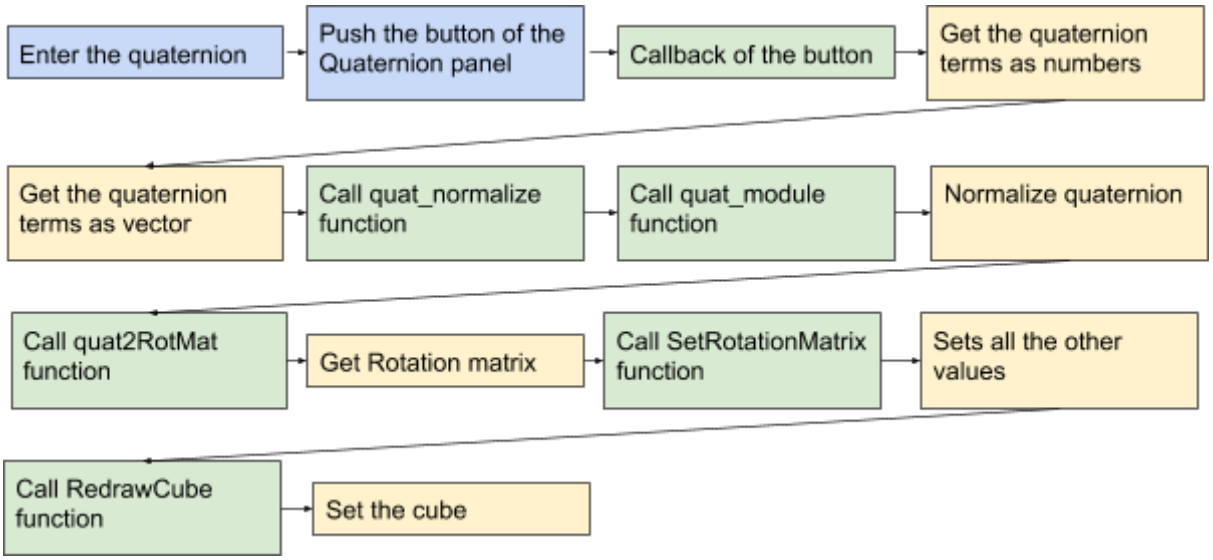
2. Euler Angle/Axis

a. Enter an angle in radians and the three terms of the axis in the editable texts.
b. Push the button of the Euler Angle/Axis' panel.
c. The callback of the button gets the angle and the three terms and get them as a vector. Then it calls the Eaa2rotMat to transform that information into a rotation matrix. Finally it calls the RedrawCube to set the cube with the new rotation matrix.



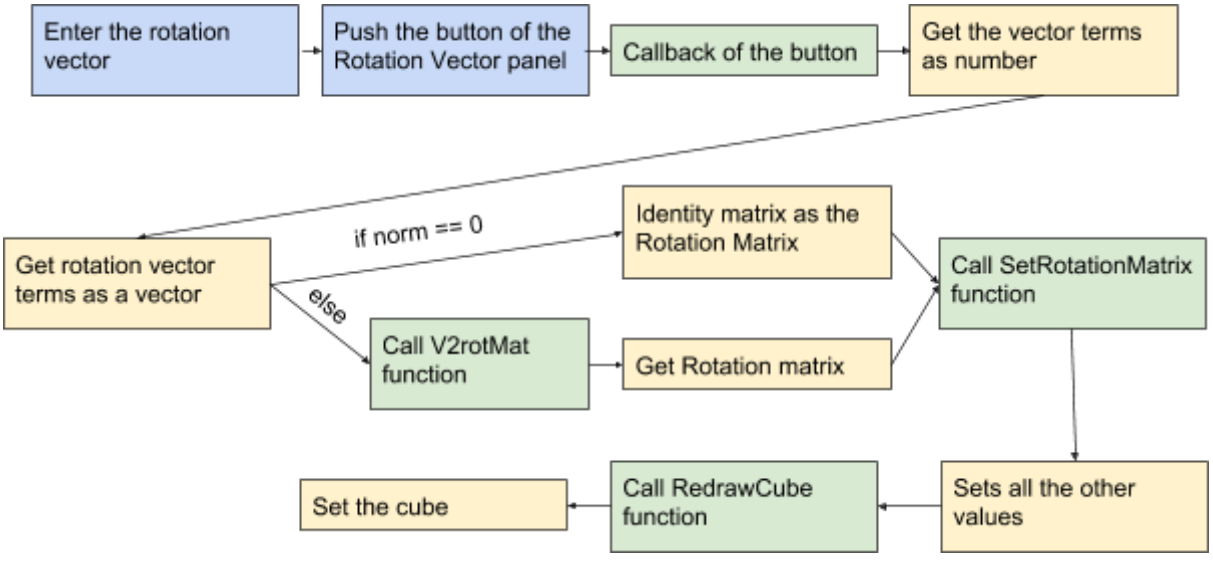
3. Quaternion

a. Enter the four terms of the quaternions in the editable texts.
b. Push the button of the Quaternion's^{o} panel.
c. The callback of the button gets the editable texts and transforms it in a quaternion; then it calls the quat_normalize function, that at the same time calls the quat_module function, to normalize the quaternion; then calls the quat2RotMat function to transform it to a rotation matrix. Finally it calls the RedrawCube to set the cube with the new rotation matrix.



4. Rotation Vector

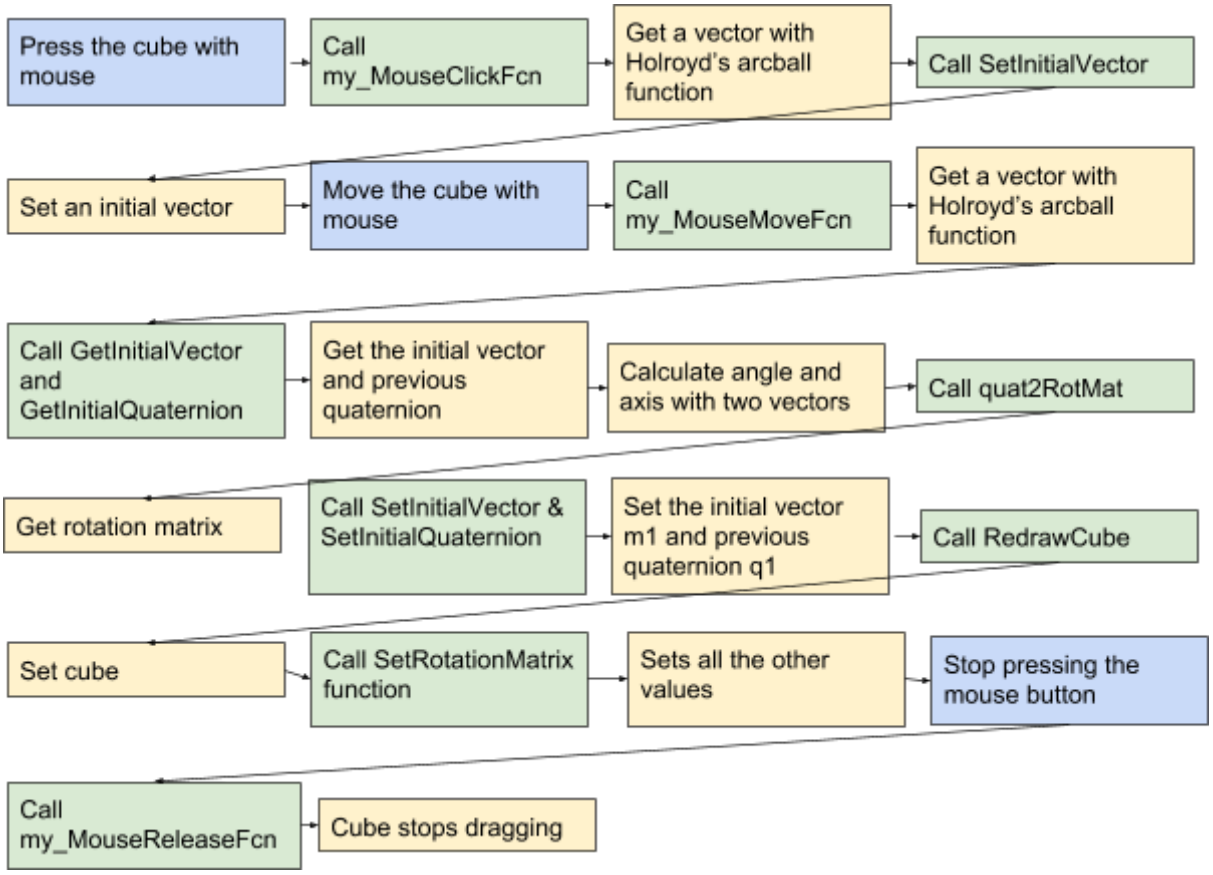
a. Enter the three terms of the Rotation Vector.
b. Push the button of the Rotation Vector's panel.
c. The callback of the button gets the three values and transform them as a vector. Then if the norm is 0 the rotation matrix is the identity matrix, if not, it calls the V2rotMat function using the vector to get the rotation matrix that it's going to be used to call the RedrawCube function.



5. Drag cube

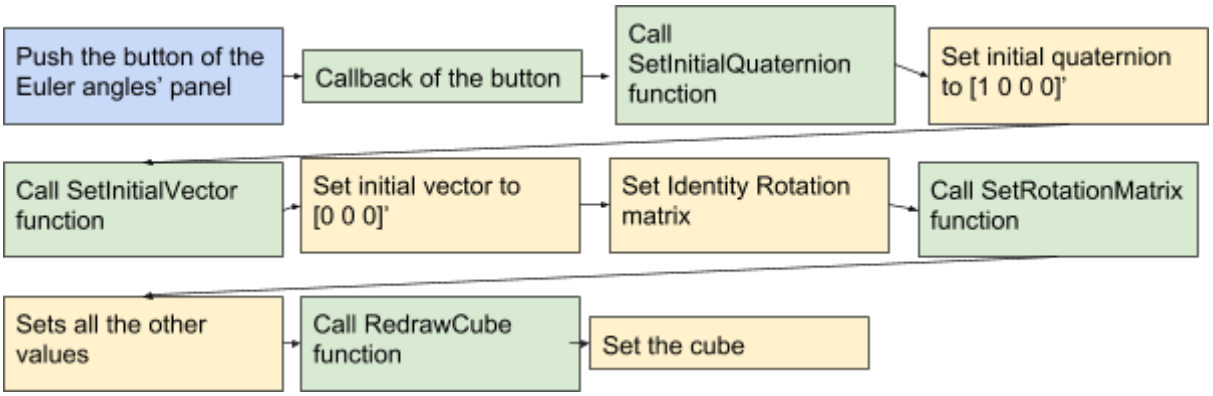
a. Press the cube with the mouse so my_MouseClickFcn is executed and sets an initial vector with SetInitialVector.

b. Move the cube with the mouse so my_MouseMoveFcn is executed, it gets the initial vector with GetInitialVector and a new vector with the Holroyd's arcball function. Uses the Eaa2rotMat to transform the axis and the angle, calculated with these two vectors, to a Rotation Matrix and calls the RedrawCube function.
c. Stop pressing the button in order to stop dragging so finally my_MouseReleaseFcn is executed and the cube stops dragging.

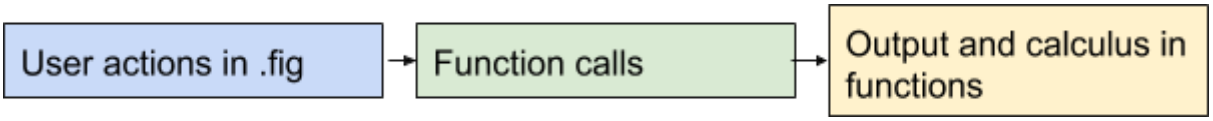


6. Reset

a. Press the reset button.
b. The callback of the button sets the edit panels to 0, then creates an identity rotation matrix and calls the RedrawCube function to reset the cube.



Diagrams' legend



Youtube video

https://www.youtube.com/watch?v=-8ozJZLvLpk&feature=youtu.be