2D Robotic Arm Simulator With Collision Avoidance

★ Favorite	☐
Status	Inbox
Notebooks	Robotic Project Description Files
Created	@July 14, 2025
Edited	@July 14, 2025 1:55 PM
Archive	☐
Pin	☐

Introduction:

Many of us may be new to robotics. Though we generally have an idea of what this field is about. If not, then consider this field to be a collection to tools that are made to reduce human effort. I would classify robotics into two sectors. The projects that reduce mental effort (Artificial Intelligence) and the ones that reduce physical effort (Robots). I focus on both of them.

To begin with this project, I will give a short “know-before-anything” information that you need in order to understand my project.

To create robots, building and designed the structure for implementation is one thing, but making it move and perform correctly is another. Structures are based on connecting links (a set of bodies) using joints. However, this project does not focus on the physical structure of the robot. This focuses on the movement of the robot.

A robot’s movement can be described with its configuration space (C-Space). The C-Space is the set of all possible configurations/positions a robot can reach in its 3D space. This can be done through a rotation followed with a translation. Typically, all movements, not just in robotics but also in real life, can be described through a rotation followed with a translation (This is known as the Chasles Theorem). As simple as it may sound, it becomes difficult when try to implement it yourself through pure mathematics. Our ancestors though would have no other option but to calculate it by hand; however, we can always automate these processes through computers. Hence, given the necessary information for calculating a robot’s next configuration, this program will perform those calculations and save your time. Now, lets begin by understanding the mathematics behind calculating the configurations of robots

The Mathematics Behind it All:

As I mentioned before, a robot may achieve any position through a rotation followed by a translation. If you have not already figured it out, than the two pieces of information we need is: how much we rotate and how much we translate. Mathematically, we can represent the amount of rotation by an angle θ in radians and the amount of translation by x in any units depending on the specification.

Although to actually perform the calculations, we need to represent the rotation of the robot in a matrix. In every case, after rotation, at least one coordinates of the 3D space (x, y, z) change. This depends on whether we rotate around one or more axes, and depending on which axis it is, there are rotation matrices that calculate the new rotated point. You are allowed to rotate the robot around multiple axes; however, a common convention is to follow the order of rotations of $Z - Y - X$. A common way to specify these axes in mathematics is Yaw(z-axis), Pitch(Y-axis), and Roll(x-axis). One important point is that the overall matrix rotation equation shall only be used when it rotates around all three axes. If it rotates around only 2 or one than only those rotation matrices shall be multiplied to obtain the overall rotation matrices.

The overall rotation matrix R is obtained through multiplying the rotation matrices of each applied axis in the common convention order:

$$R = R_z(\phi) \times R_y(\theta) \times R_z(\psi)$$

The angles ϕ, θ, ψ are the rotations of the robot performed around each axis. Now lets perform necessary calculations to obtain the general form of the rotation matrix.

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$R_{zyx} = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi)$$

$$R = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

This covers the first part of a robot’s change in configuration which is its rotation. The next part shall cover its translation, which is much simpler compared to the process of rotation. Once you have a point in 3D space you multiply it with the rotation matrix and then add the translation vector. Now that we have covered the mathematics of it all. Lets see how it has been implemented in programs to automate the process.

The Programming Aspect:

To program these calculations we will use C++ as our language. To build our program we divide our files into three sections: include, src, and CMakeLists.txt.

Before we begin anything, we must have our Eigen library installed because it makes matrix calculations, like the ones you saw above, a piece of cake. To install this library, run this command on your terminal.

```
git clone https://github.com/microsoft/vcpkg.git
cd vcpkg

.\bootstrap-vcpkg.bat

.\vcpkg integrate install
```

This installs the vcpkg package manager. In case of those who don't know, vcpkg is package manager for C++ that makes the process of installing libraries like Eigen easier.

Once vcpkg is installed, go ahead and run the below command to install the Eigen library:

```
# Install Eigen (header-only, no compilation needed)
vcpkg install eigen3
```

As I mentioned before, we would also need to create the CMakeLists.txt. The purpose of creating such a file is simple: CMake is a meta-build system that helps run your projects on other OS systems easily. If this code were to run on another compiler or OS, than CMake would help adapt to that process .The other benefits of such a file is:

1. Cross-Platform Support: It works on Windows, Linux, and MacOS which simplifies having to create a different CMake for every OS System.

2. Dependency Management: It helps to easily link libraries like eigen using “find_package()”
3. Automate Builds: It helps compile with a single command.

Once you have created your CMake file, go ahead an copy the below content:

```
cmake_minimum_required(VERSION 3.15)
project(RoboticArmSimulator)

# Find Eigen
find_package(Eigen3 REQUIRED)

# Add executable
add_executable(arm_simulator
    src/main.cpp # These are the name of the files that I have created, you are free to
    src/transform2d.cpp
    src/robot_arm.cpp
)

# Link Eigen
target_link_libraries(arm_simulator PRIVATE Eigen3::Eigen)

# Include directories
target_include_directories(arm_simulator PRIVATE
    ${EIGEN3_INCLUDE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/include
)
```

Now for our actual programs, I have divided the implementations into three files: main.cpp, robot arm, and transform2d.

Here is the below code for the robot_arm.cpp

```
#include "robot_arm.hpp"
#include <cmath>
```

```
std::vector<Transform2D> RobotArm::getArmPose(
    const std::vector<double>& joint_angles
) const {
    std::vector<Transform2D> poses;
    Transform2D current_pose; // Initialized to zeros via class definition

    for (size_t i = 0; i < joint_angles.size(); ++i) {
        // Update orientation
        current_pose.theta = (i == 0) ? joint_angles[i]
                              : poses.back().theta + joint_angles[i];

        // Update position (skip for first joint)
        if (i > 0) {
            double prev_len = link_lengths[i-1];
            current_pose.x = poses.back().x + prev_len * std::cos(poses.back().theta);
            current_pose.y = poses.back().y + prev_len * std::sin(poses.back().theta);
        }

        poses.push_back(current_pose);
    }
    return poses;
}
```

Here is the below code for transform2d.cpp

```
#include "transform2d.hpp"
#include <Eigen/Dense>
#include <cmath>
#include <limits>

Eigen::Matrix3d Transform2D::matrix() const {
    Eigen::Matrix3d T;

    // Clean near-zero values
    auto clean = [](double val) {
        const double epsilon = 10 * std::numeric_limits<double>::epsilon();
        return std::abs(val) < epsilon ? 0.0 : val;
    };
```

```
    const double c = clean(std::cos(theta));
    const double s = clean(std::sin(theta));

    T << c,    -s,    clean(x),
         s,     c,    clean(y),
         0.0,   0.0,  1.0;

    return T;
}

Eigen::Vector2d Transform2D::operator*(const Eigen::Vector2d& point) const {
    Eigen::Vector3d homog_point(point.x(), point.y(), 1.0);
    Eigen::Vector3d transformed = matrix() * homog_point;

    // Final cleanup of results
    auto clean = [](double val) {
        return std::abs(val) < 1e-10 ? 0.0 : val;
    };

    return {clean(transformed.x()), clean(transformed.y())};
}
```

Here is the below code for robot_arm.hpp

```
#pragma once
#include "transform2d.hpp"
#include <vector>

class RobotArm {
public:
    std::vector<double> link_lengths;

    explicit RobotArm(const std::vector<double>& lengths)
        : link_lengths(lengths) {}

    std::vector<Transform2D> getArmPose(const std::vector<double>& joint_angles) c
};
```

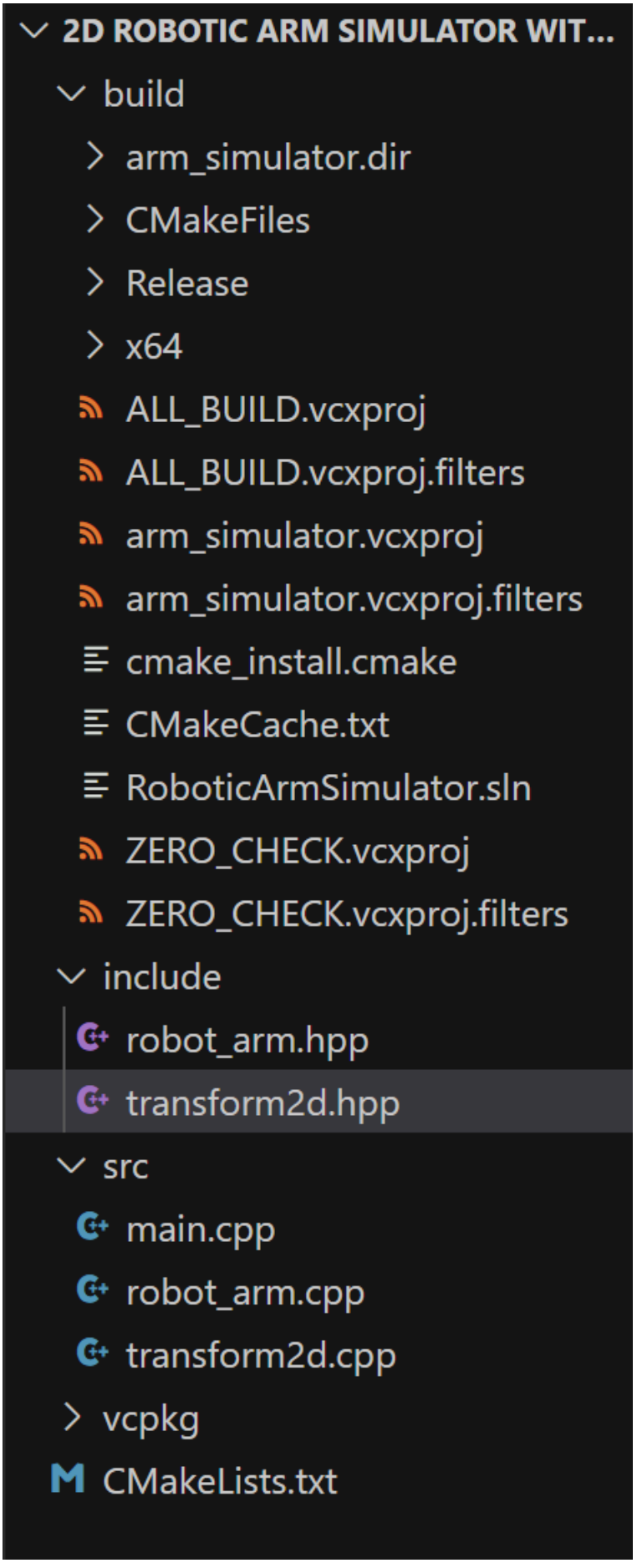
Here is the below code for transform2d.hpp

```
#pragma once
#include <Eigen/Dense>

class Transform2D {
public:
    double x = 0.0;       // Initialize to zero
    double y = 0.0;       // Initialize to zero
    double theta = 0.0;   // Initialize to zero

    // Generate the 3×3 transformation matrix
    Eigen::Matrix3d matrix() const;

    // Transform a 2D point (operator overloading)
    Eigen::Vector2d operator*(const Eigen::Vector2d& point) const;
};
```

Here is the run command I used to run this program:

```
cmake --build build --config Release
>> .\build\Release\arm_simulator.exe
```

For reference, I have included a photo of my file sections:


2D ROBOTIC ARM SIMULATOR WIT...
build
arm_simulator.dir
CMakeFiles
Release
x64
ALL_BUILD.vcxproj
ALL_BUILD.vcxproj.filters
arm_simulator.vcxproj
arm_simulator.vcxproj.filters
cmake_install.cmake
CMakeCache.txt
RoboticArmSimulator.sln
ZERO_CHECK.vcxproj
ZERO_CHECK.vcxproj.filters
include
robot_arm.hpp
transform2d.hpp
src
main.cpp
robot_arm.cpp
transform2d.cpp
vcpkg
CMakeLists.txt