Implementing the Builder Design Pattern in an Autonomous Car Vision System

Frida Cano Falcón

Java Backend Academy MTY
August 2024

Week 3

August 30, 2024

Introduction

The *Builder* design pattern is a creational pattern in software engineering that simplifies the construction of complex objects step by step. It is particularly useful when an object needs to be created by combining several optional components or when the construction process involves multiple steps. The pattern provides a flexible solution to instantiate objects, allowing developers to separate the construction of an object from its representation.

In the context of my work in my last year of college on a Self-Driving Vehicle project, I specialize in the development of perception algorithms. These algorithms are essential for processing and interpreting the visual data captured by the vehicle's camera, enabling the vehicle to make informed decisions while navigating. Given the complexity and modular nature of these perception algorithms, the *Builder* pattern offers an efficient way to manage and organize the various stages of image processing within the autonomous vehicle's vision system.

This report details the implementation of the Builder design pattern in a Java 17 project that simulates the image processing pipeline of an autonomous car's perception system. The project demonstrates how the Builder pattern can be applied to create a flexible and extensible image processing pipeline.

Project Overview

The project consists of four main classes:

1. *ImageProcessingStage*
2. *ImageProcessingPipeline*
3. *ImageProcessingPipelineBuilder*
4. *VisionSystem*

Each class plays a role in constructing and executing the image processing pipeline, which consists of multiple stages such as image loading, preprocessing, object detection, pedestrians detection and lane detection.

1. ImageProcessingStage.java

The *ImageProcessingStage* class represents a single stage of the image processing pipeline. Each stage has a name and associated parameters. This class encapsulates the details of a specific image processing operation.

src > com > autonomouscar > vision > ImageProcessingStage.java > ImageProcessingStage

```java
package com.autonomouscar.vision;

public class ImageProcessingStage {
    private String stageName;
    private String parameters;

    public ImageProcessingStage(String stageName, String parameters) {
        this.stageName = stageName;
        this.parameters = parameters;
    }

    @Override
    public String toString() {
        return "Stage: " + stageName + ", Parameters: " + parameters;
    }
}
```

2. ImageProcessingPipeline.java

The *ImageProcessingPipeline* class holds the list of stages created by the builder and provides a method to execute the entire pipeline. Each stage is executed in sequence, simulating the real-world image processing tasks in an autonomous vehicle.

src > com > autonomouscar > vision > ImageProcessingPipeline.java > ImageProcessingPipeline

```java
package com.autonomouscar.vision;

import java.util.List;

public class ImageProcessingPipeline {
    private List<ImageProcessingStage> stages;

    public ImageProcessingPipeline(List<ImageProcessingStage> stages) {
        this.stages = stages;
    }

    public void execute() {
        stages.forEach(stage -> System.out.println(stage));
    }
}
```

3. ImageProcessingPipelineBuilder.java

The *ImageProcessingPipelineBuilder* class is responsible for constructing the *ImageProcessingPipeline*. It allows the user to add various stages to the pipeline using the *addStage()* method and then create the complete pipeline with the *build()* method.

src > com > autonomouscar > vision > J ImageProcessingPipelineBuilder.java > ...

```java
package com.autonomouscar.vision;

import java.util.ArrayList;
import java.util.List;

public class ImageProcessingPipelineBuilder {
    private List<ImageProcessingStage> stages = new ArrayList<>();

    public ImageProcessingPipelineBuilder addStage(String stageName, String parameters) {
        stages.add(new ImageProcessingStage(stageName, parameters));
        return this;
    }

    public ImageProcessingPipeline build() {
        return new ImageProcessingPipeline(stages);
    }
}
```

4. VisionSystem.java

The *VisionSystem* class is the main class that demonstrates how the *Builder* pattern is used to create and execute an image processing pipeline.

src > com > autonomouscar > vision > J VisionSystem.java > ...

```java
package com.autonomouscar.vision;

public class VisionSystem {
    Run | Debug
    public static void main(String[] args) {
        ImageProcessingPipelineBuilder builder = new ImageProcessingPipelineBuilder();

        ImageProcessingPipeline pipeline = builder
            .addStage(stageName:"Image Loader", parameters:"multisense_camera")
            .addStage(stageName:"Preprocessing Filter", parameters:"GaussianBlur")
            .addStage(stageName:"Object Detector", parameters:"YOLOv8")
            .addStage(stageName:"Pedestrian Detector", parameters:"YOLOv8")
            .addStage(stageName:"Lane Segmentation", parameters:"HoughTransform")
            .build();

        pipeline.execute();
    }

}
```

Conclusion

The *Builder* design pattern provides a powerful and flexible way to manage the construction of complex objects in software systems. In this project, the pattern was applied to create an extensible image processing pipeline for an autonomous car's vision system. By breaking down the pipeline into distinct stages and allowing them to be built in a modular fashion, the Builder pattern enhances the maintainability and scalability of the system.

This implementation showcases how design patterns like Builder can be effectively used in real-world applications, particularly in fields like robotics and autonomous systems, where complex and configurable processes are common.