Below is a **high-level pipeline** you might design to **evaluate** autonomous parking maneuvers and **detect stuck states**. It’s inspired by common techniques in autonomous driving (similar to Tesla’s approach) but generalized for any parking‐lot scenario.

## 1. Data Collection & Logging

* **Sensors**: Camera (front/rear), ultrasonic, LiDAR (if available), wheel encoders, IMU.
* **CAN Bus / Vehicle Logs**: Steering angle, throttle commands, braking, etc.
* **Environment / Map Data**: (If using a pre-mapped garage or known layout.)

**Goal**: Ensure consistent, **timestamped** multi-sensor data for each parking maneuver.

## 2. Data Preprocessing

1. **Time Synchronization**:
   * Align sensor timestamps (camera frames, ultrasonic pulses, etc.) with vehicle logs.
2. **Sensor Fusion / Calibration**:
   * Calibrate and/or fuse ultrasonic, camera, and inertial data into a **common coordinate frame**.
3. **Noise Reduction**:
   * Apply filters (e.g., a low-pass filter on IMU data, noise reduction on ultrasonic pulses).

**Goal**: Produce a **clean, unified** data stream that accurately represents the vehicle’s motion and environment.

## 3. Feature Extraction

From the preprocessed data, derive **key signals** you need to detect “stuck” states:

**Vehicle Kinematics**

* + Velocity, acceleration, steering angle, yaw rate (from wheel encoders + IMU).
  + Compare commanded speed vs. actual speed to see if the car is failing to move as expected.

**Obstacle Proximity**

* + Distance readings from ultrasonics or bounding boxes from camera-based object detection.
  + Track changes in distance over time.

**Occupancy / Free Space Metrics** (if using LiDAR or advanced vision)

* + Calculate a local **occupancy grid** or free space in front/behind the car.
  + If occupancy remains unchanged despite movement commands, suspect a stuck scenario.

**Goal**: Produce numerical or categorical features that feed directly into your stuck-state evaluation logic.

## 4. Define “Stuck” Criteria

Formally specify conditions that qualify as “stuck.” Examples:

**Threshold-based**

* + Velocity < 0.1 m/s for more than X seconds, with > Y% throttle.
  + Ultrasonic or camera-based detection indicates an unchanging obstacle within 20 cm for > T seconds.

**Model-based / Learning-based**

* + Train a **classifier or anomaly detector** (e.g., an SVM or small neural network) on real logs labeled as “stuck” vs. “not stuck.”
  + Input features: sensor mismatch, velocity profiles, object distance trends, etc.

**Hybrid Approach**

* + Start with threshold-based heuristics.
  + Use a learning model to refine borderline cases, reduce false positives/negatives.

**Goal**: Translate “stuck” from a vague concept into **quantifiable** parameters or a trained classifier.

## 5. Online Detection Logic

In real-time (or in replay analysis):

1. **Compare** the “planned path” or velocity commands to **actual** motion.
2. **Evaluate** your extracted features against the “stuck” criteria.
3. **Accumulate Evidence** (e.g., require the condition to hold for a certain duration) to avoid false triggers.

**Goal**: Raise a “stuck” flag once you cross certain thresholds or your classifier’s confidence is sufficiently high.

## 6. Trigger Actions Upon Detection

Once a stuck state is confirmed:

1. **Re-Plan**
   * Attempt a new path around obstacles (slight steering changes, forward-backward “wiggling”).
2. **Increase Sensor Attention**
   * Re-check surroundings at higher frequency or broader angle.
3. **Alert the Driver**
   * If the system can’t resolve automatically, issue an alert or request human intervention.
4. **Failsafe Stop**
   * If re-planning fails repeatedly, halt the vehicle in a safe orientation.

**Goal**: Safely recover from or handle the stuck state.

## 7. Testing & Validation

1. **Simulation Tests**
   * Construct large-scale synthetic scenarios with different obstacles, inclines, lighting conditions, etc.
   * Evaluate accuracy (true positives, false positives) and overall pipeline performance.
2. **Real-World Garage Trials**
   * Gather “stuck-state” examples with ground-truth labeling.
   * Log success rate, recovery time, or driver intervention frequency.

**Goal**: Ensure reliability across diverse real-world conditions.

## 8. Iteration & Continuous Improvement

* **Collect Feedback** from each parking attempt or user report.
* **Refine** threshold parameters or retrain the stuck-state classifier.
* **Expand** scenario coverage (different garage layouts, presence of pedestrians, etc.).

**Goal**: Evolve and maintain the system for **robust** and **scalable** performance.

### ****Summary****

An effective stuck‐state detection system uses:

1. **Robust sensor fusion** to accurately know the vehicle’s environment and motion.
2. **Clear definitions** of what “stuck” means (threshold- or model-based).
3. **Real-time detection** with subsequent re-planning or alerts.
4. **Extensive validation** in both simulation and actual parking garages.

Building this pipeline involves **both** a data‐driven approach (collecting/labeling real stuck events) and **traditional robotics algorithms** (planning, sensor fusion). This ensures your system can detect and handle corner cases—literally—in a garage setting.

### ****How to Integrate “Stuck-State Detection Accuracy” Metric into the System****

To effectively integrate **stuck-state detection accuracy (precision/recall/F1-score)** into the reversing pipeline, you need to:

1. **Define Ground Truth** (Labeling Stuck Events)
2. **Implement Real-Time Stuck Detection in the System**
3. **Log Predictions and Compare with Ground Truth**
4. **Compute Precision, Recall, and F1-Score**
5. **Automate the Evaluation & Continuous Monitoring**

## ****1. Define Ground Truth (Labeling Stuck Events)****

Before measuring accuracy, you need **ground truth**—a reliable label indicating whether a vehicle was actually stuck. This can be obtained through:

* **Human Annotation**
  + Review past test logs and label instances where the car was **truly stuck** (e.g., didn’t progress despite movement commands).
  + Use video playback and sensor data (speed, steering, distance readings).
* **Automated Labeling with Rules** (to speed up annotation)
  + Define simple **heuristics** to auto-label events that likely indicate being stuck:
    - Speed < 0.1 m/s for **X seconds**
    - Continuous steering oscillation with no forward progress
    - Obstacle distance unchanged for **Y seconds**

Once the **ground truth dataset** is ready, move on to system integration.

## ****2. Implement Real-Time Stuck Detection****

In your reversing **software stack**, create a **stuck-state detection module** that runs in real time.

**Input Features:**

* + Speed, acceleration, throttle/brake commands, steering angle, obstacle distance.
  + Sensor fusion (camera, ultrasonic, IMU) to detect when the car **fails to move** as expected.

**Detection Algorithm:**

* + **Hybrid approach** (as discussed earlier):
    1. **Threshold-based checks** (speed, distance) for quick flags.
    2. **SVM-based classifier** for borderline cases (reduces false positives).
    3. **Time window verification** (e.g., stuck must persist for > 3 sec).

**System Output:**

* + Every **frame**, the system classifies:
    - **0** → Not stuck
    - **1** → Stuck

This prediction stream is logged alongside **ground truth**.

## ****3. Log Predictions & Compare with Ground Truth****

To evaluate detection accuracy, the system must **log**:

1. **Stuck-state prediction output** from real-time software
2. **Ground truth labels** (from annotated test data)
3. **Timestamps** to align logs for analysis

Each test **frame-by-frame**, you can now **compare predictions vs. actual labels**.

## ****4. Compute Precision, Recall, F1-Score****

Using **true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN)**, compute:

* **Precision** = TP / (TP + FP)  
  (Out of all times the system declared “stuck,” how many were actually stuck?)
* **Recall** = TP / (TP + FN)  
  (Out of all actual stuck cases, how many did the system detect?)
* **F1-Score** = Harmonic mean of precision & recall

### ****Example Calculation****

| **Frame #** | **Ground Truth (GT)** | **Prediction (P)** |
| --- | --- | --- |
| 1 | 0 (Not Stuck) | 0 (Not Stuck) ✅ TN |
| 2 | 1 (Stuck) | 1 (Stuck) ✅ TP |
| 3 | 1 (Stuck) | 0 (Not Stuck) ❌ FN |
| 4 | 0 (Not Stuck) | 1 (Stuck) ❌ FP |
| 5 | 1 (Stuck) | 1 (Stuck) ✅ TP |

* **Precision** = 2 / (2 + 1) = **66.7%**
* **Recall** = 2 / (2 + 1) = **66.7%**
* **F1-Score** = **66.7%**

## ****5. Automate Evaluation & Continuous Monitoring****

**Post-Test Evaluation:**

* + After each test run, log **predictions vs. ground truth**, auto-calculate **F1-score**, and push results to a dashboard.

**Continuous Monitoring in Production:**

* + Deploy real-time monitoring:
    - If false positives are too high, tweak thresholds or SVM model.
    - If false negatives increase, adjust stuck criteria.

### ****Final Integration Flow****

1. **Run Test Drive:** Car performs reversing maneuvers in a controlled garage.
2. **System Detects Stuck Events:** Logs predictions in real-time.
3. **Compare to Ground Truth:** Label whether car was actually stuck or not.
4. **Compute Metrics (Precision, Recall, F1-Score).**
5. **Improve Model:** Adjust SVM parameters, refine thresholds, re-train if needed.

### ****Outcome****

The **stuck-state detection module** becomes: ✅ **Quantifiable** (F1-score tracks detection quality).  
✅ **Optimizable** (If precision or recall drops, we refine it).  
✅ **Deployable** (Real-time + offline validation ensures consistency).

### ****Summary****

To **integrate** the **stuck-state detection accuracy metric**, you:

1. **Define and collect ground truth data** (manually or auto-labeled).
2. **Run real-time stuck detection** (thresholds + SVM classifier).
3. **Log predictions & compare with ground truth** (frame-by-frame).
4. **Compute precision, recall, and F1-score** for evaluation.
5. **Automate and continuously monitor** to improve detection over time.

This structured approach ensures **continuous iteration**, making the reversing feature more reliable in complex garage scenarios.