IRIS
Iris Dynamics Ltd.

TECHNICAL NOTE 260423

Force Feedback Labelling Application
Utilizing Haptic Mode for Fast Motion and Gentle Contact

Revision 260424
irisdynamics.com | +1 (888) 995 -7050

Introduction

Automated labeling applications demand fast motion, precise positioning, and gentle contact with the product surface. ORCA™ Series Smart Linear Motors combine high-speed linear actuation with built-in force control, allowing machines to transition seamlessly from rapid approach to compliant pressing. This enables reliable and repeatable label application while reducing mechanical complexity and improving overall system performance.

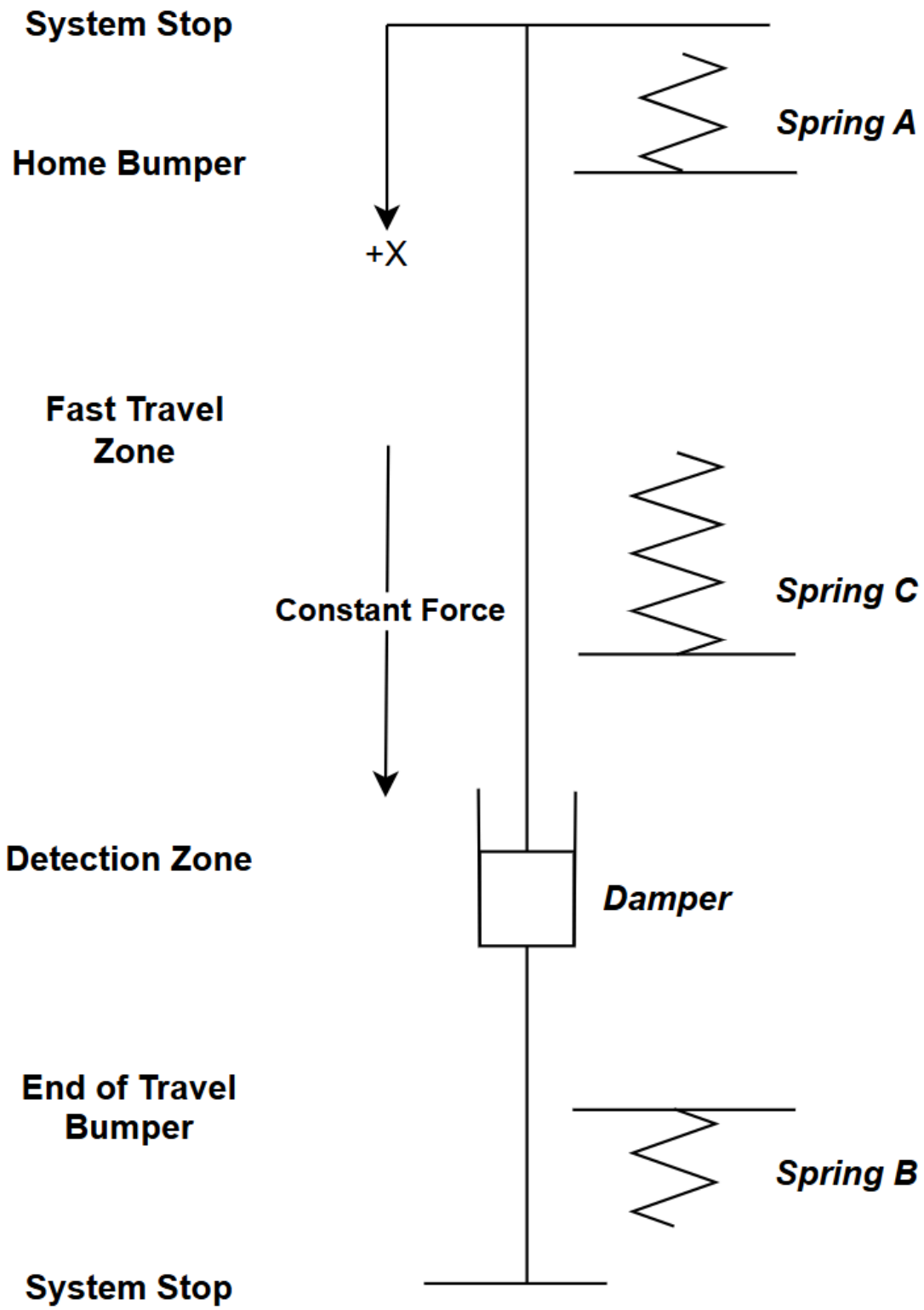
A similar approach can also be used in other applications that involve pressing such as stamping or processes that require applying a specified force for a defined duration.

ORCA motors provide several modes of operation optimized for different use cases. The presented application makes use of **Haptic Mode**, which allows multiple haptic effects (springs, dampers, oscillations) to be layered together to create complex dynamic behavior. This mode is particularly effective in applications where there is variability in position or timing. See the Haptic Mode Tutorial to get familiar with this mode.

The example described here uses an ORCA motor in an application requiring **fast motion (high cycle time)** while simultaneously maintaining **gentle contact with the package surface**.

The system operates as follows:

1. The shaft begins at a home position, resting on a **Home Bumper**
2. When triggered, it moves rapidly through a **Fast Travel Zone** where no contact is expected.
3. The system transitions into a controlled approach in the **Detection Zone**
4. When the shaft contacts the package, a defined force is applied, referred to as the *apply_force*.
5. The system either holds the force for a specified time or immediately returns home.
6. If the system reaches the **End of Travel Bumper** without a collision, immediately return home



Haptic Effects Overview

Spring A - Home Bumper

This is configured as a high-gain single-direction spring used to slow the shaft near the home position.

This soft stop prevents the shaft from striking physical stops such as shaft collars or other mechanical components.

Spring A defines a stable equilibrium at the home position, minimizing the power required to hold the shaft in place.

Spring B - End of Travel Bumper

This spring acts as a soft stop at the opposite end of travel. If the motor cycles without a package present, **Spring B** prevents the shaft from contacting mechanical hard stops.

Spring C - Rapid Move

This spring is used to create the fast movement, slingshotting the shaft through the fast travel zone. This spring will be used to implement a section of travel with a constant force, creating a controlled acceleration region. This is done by using a very high gain and setting a saturation value. The saturation value becomes a system tuning parameter to determine how fast the shaft accelerates through the fast travel zone. The spring is one sided and ends where deceleration to the controlled approach speed will begin.

Damper - Speed Control

The damper effect interacts with the applied **Constant Force** to determine the steady-state approach speed.

The damper gain combines with the applied force according to:

$$Speed \ * \ damper\_gain \ = \ apply\_force \ + \ F_{gravity}$$

This relationship determines the constant approach velocity.

Constant Force - Apply/Return Trigger

The **Constant Force** effect is used at runtime to trigger transitions between system states.

Apply State

Set the **Constant Force** to the desired detect or pressing force.

Return Home

Set the **Constant Force** to a large negative value to retract the shaft rapidly.

System States

The actuator cycle uses two primary states in addition to the initial auto-zero procedure during startup.

- RETURN_HOME
- APPLY

RETURN_HOME State

This state retracts the shaft and also acts as the idle state while waiting for the next cycle trigger.

In this state:

- **Constant Force** is set to a large negative value.
- The shaft accelerates upward toward the home position.
- Motion continues until the shaft enters the region defined by **Spring A**.

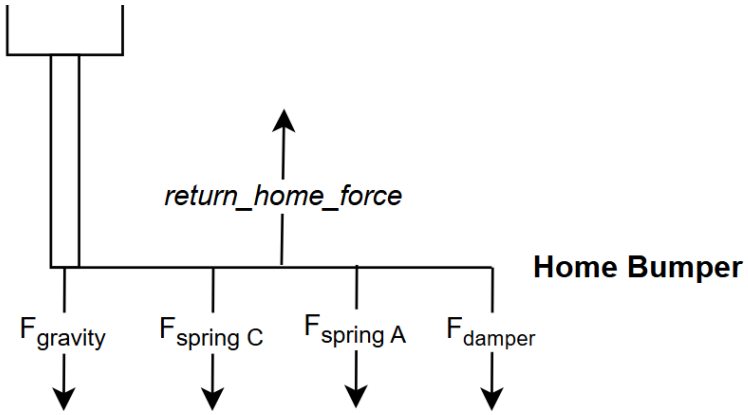
Spring A then compresses until equilibrium is reached when:

$$Spring\ A\ Force\ =\ return\_home\_force\ -\ F_{gravity}$$

For *motor_position < spring_A_position*

$$Spring\ A\ Force\ =\ spring\_A\_gain\ *\ (motor\_position\ -\ spring\_A\_position)$$

The system stabilizes at this point, resting at the home position.



The *return_home_force* must be large enough to overpower both the damper and **Spring C** while in the **Fast Travel Zone.**

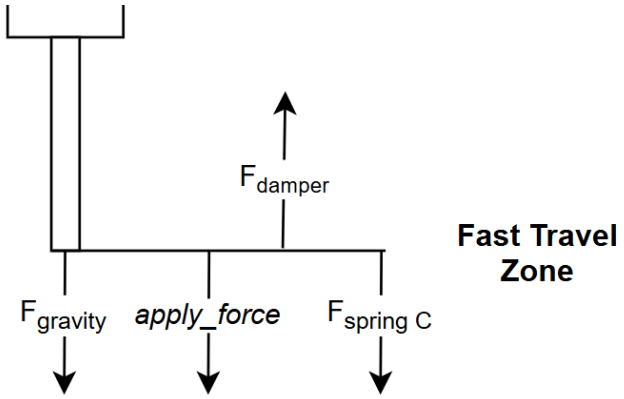
APPLY State

In the APPLY state, the **Constant Force** is switched from the *return_home_force* to *apply_force*.

At this point:

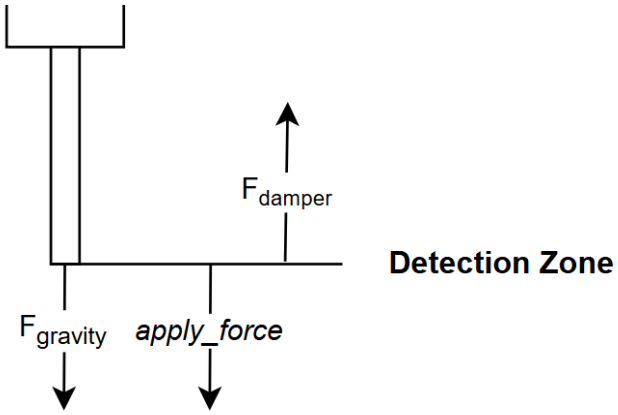
- **Spring A** releases
- **Spring C** accelerates the shaft downward
- Gravity assists the motion

This produces a rapid movement through the fast travel zone.



When the shaft reaches the end of **Spring C**, the remaining dynamics are:

- system inertia
- force due to gravity
- *apply_force*
- damper force



The damper produces a steady approach speed where:

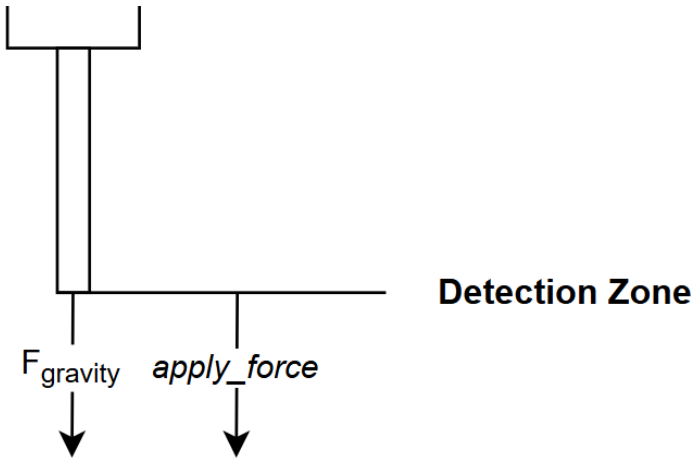
$$speed = \frac{apply\_force + F_{gravity}}{damper\_gain}$$

The shaft continues downward at this constant speed until it contacts the package.

At contact:

- Shaft velocity drops to zero
- Damping force disappears
- Only *apply_force + gravity* remains

This becomes the applied pressing force.



Once contact is detected, the system can:

- Hold the force for a specified duration
- Wait for an external trigger
- Immediately return home
- Rest at the bottom position without applying force by setting the *apply_force* = gravity

Detection

Reliable detection cannot rely solely on force measurements.

Instead, detection should use a combination of conditions:

- Speed = 0
- Force = *apply_force*
- Position < **Spring B**

When these conditions are met, the system can confidently determine that contact with the package has occurred.

This approach is robust and resistant to noise.

The system can also detect empty cycles when:

- Speed = 0
- Force ≈ negative $F_{gravity}$
- Position >= **Spring B**

This indicates the shaft reached the end of travel without encountering a package.

Tuning Notes

Equilibrium Home Position

In this the force due to **Spring C** will be equal to its saturation value due to the high spring gain.

$$home\_position = \frac{F_{Spring\ C} + F_{gravity} - return\_home\_force}{spring\_A\_gain} - spring\_A\_position$$

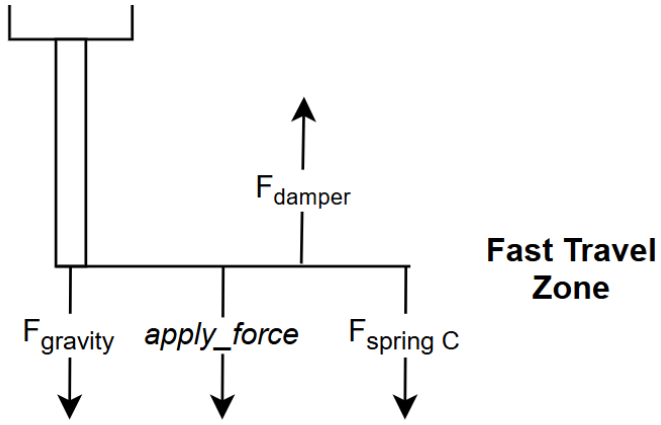
Deceleration Distance

Increasing *spring_C_position* -> increase distance to reach target approach speed
Decrease *spring_C_position* -> decrease distance to reach target approach speed

Acceleration Force

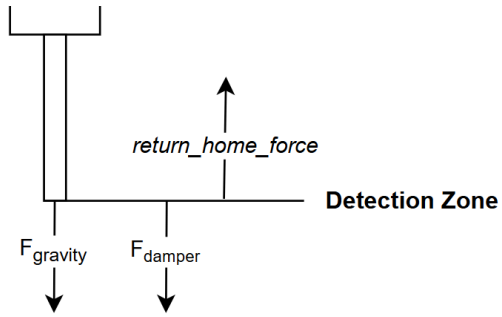
Where the moving mass is a combination of the shaft and any attached components..

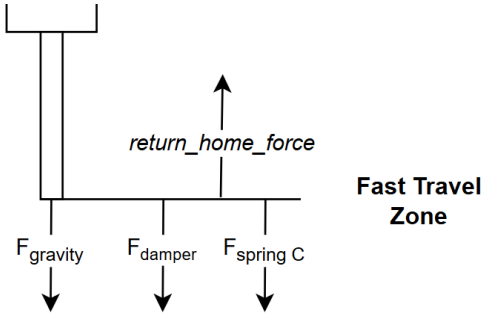
$$moving\ mass * acceleration = F_{Spring\ C} + F_{gravity} + apply\_force - speed * damping\_gain$$



Retract Speed

Retract speed depends on multiple effects. However, by selecting a sufficiently large *return_home_force*, other effects (aside from **Spring A**) can be dominated.



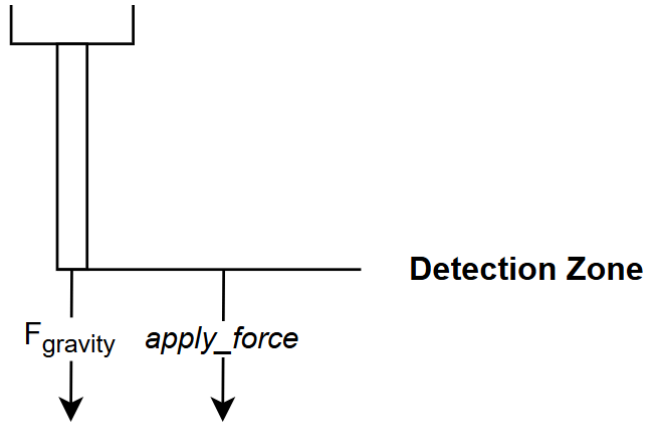


Approach Speed:

$$speed = \frac{apply\_force + F_{gravity}}{damper\_gain}$$

Applied Force:

$$force\ applied\ to\ package = apply\_force + F_{gravity}$$



Note:

- The motor reports only the force it generates.
- The actual force applied to the package includes the weight of the shaft.

Initial impact force may also include inertial effects which are not directly measurable.

Communication

IrisControls

IrisControls is an accessible interface for testing and tuning purposes, which allows haptic effects to be configured and states to be altered by changing the **Constant Force** effect value.

PLC / Microcontroller

All motor functionality is also available through reading and writing to the ORCA motor's Modbus registers. Configuration can be done and saved to the motor or done at the start of a program.

Then during runtime the following registers should be monitored:

- Position (µm) (342 low register, 343 high register)
- Speed (mm/s) (344 low register, 345 high register)
- Force (mN) (348 low register, 349 high register)

*Note these are all double wide registers and must be combined to a signed 32 bit integer

To control the state change, write to:

- Constant force (mN) (642 low register, 643 high register)

orcaSDK

The orcaSDK provides a direct function for configuring haptic effects and state control. An example program file is available.

C++ orcaSDK Example Code

```C/C++
#include <iostream>
#include "actuator.h"
#include <chrono>

using namespace orcaSDK;

#define STOPPED_VELOCITY 10
#define FORCE_DETECTION_ENVELOPE 2000

enum class ControlState{
    AUTO_ZEROING,
    RETURN_HOME,
    APPLY,
    HOLD_CONTACT
};
```

```
int serial_port = 8; //rs422 serial port,
ControlState current_state;

int32_t gravity = 20000; //weight of shaft mN
int32_t return_home_force = -300000;
int32_t apply_force = 40000 - gravity; //force applied to packaged 40N, gravity =
20 N
uint32_t cycle_time_ms = 3000;
uint32_t hold_contact_time_ms = 1000;
uint32_t max_apply_time = 10000;

//Haptic Effect Parameters
int32_t spring_A_gain = 10000;
int32_t spring_A_position = 15000;

int32_t spring_B_gain = 10000;
int32_t spring_B_position = 200000;

int32_t spring_C_gain = 10000;
int32_t spring_C_position = 70000;
int32_t spring_C_saturation = 250;

int16_t damper_gain = 400;

//Timers to track time in each state
std::chrono::steady_clock::time_point cycle_start_time;
std::chrono::steady_clock::time_point apply_start_time;
std::chrono::steady_clock::time_point hold_contact_start_time;

//Motor parameters to monitor
int32_t detected_velocity;
int32_t detected_force;
int32_t motor_position;

orcaSDK::Actuator motor{ "MotorName" };

/*
* Write initial haptic configuration values to motor to set up springs and dampers
*/
void configureHapticEffects() {
```

```
    motor.set_constant_force(return_home_force);
    motor.set_spring_effect( //Home Bumper
        0,                      //Spring ID (0, 1, or 2)
        spring_A_gain,         //Spring strength
        spring_A_position, //Center position "resting retracted point"
        0,                  //Dead Zone
        0,                  //Saturation
        ORCAReg::Sn_COUPLING_Values::NEGATIVE //Spring coupling
    );

    motor.set_spring_effect( //End of travel bumber
        1,                      //Spring ID (0, 1, or 2)
        spring_B_gain,         //Spring strength
        spring_B_position, //Center position
        0,                  //Dead Zone
        0,                  //Saturation
        ORCAReg::Sn_COUPLING_Values::POSITIVE //Spring coupling
    );

    motor.set_spring_effect( //rapid move
        2,                      //Spring ID (0, 1, or 2)
        spring_C_gain,         //Spring strength
        spring_C_position, //Center position
        0,                  //Dead Zone
        spring_C_saturation,//Saturation
        ORCAReg::Sn_COUPLING_Values::NEGATIVE //Spring coupling
    );

    motor.set_damper( //speed control
        damper_gain
    );
}

/*
* Get millisecond helper function
*/
auto ms = [](auto t)
    {
        return std::chrono::duration_cast<std::chrono::milliseconds>(t).count();
    };
```

```
/*Check parameters to detect for a package contact
* force ~= apply force
* speed ~= 0
* position < end of travel bumper
*/
bool detect_contact(){
    return ((std::abs(detected_force - apply_force) <= FORCE_DETECTION_ENVELOPE)
&&
            (std::abs(detected_velocity) < STOPPED_VELOCITY) &&
            (motor_position < spring_B_position));
}

/* Check for cycle without package
* force ~= -gravity
* speed ~= 0
* position >= end of travel bumper
*/
bool detect_empty(){
    return ((std::abs(detected_force + gravity) <= FORCE_DETECTION_ENVELOPE) &&
        (std::abs(detected_velocity) < STOPPED_VELOCITY) &&
        (motor_position >= spring_B_position));
}

/*
* On state enter do initial actions, set timers to track state duration
*/
void enterState(ControlState newState) {
    current_state = newState;
    switch (newState) {
    case ControlState::RETURN_HOME:
        motor.set_constant_force(return_home_force);
        cycle_start_time = std::chrono::steady_clock::now();
        break;
    case ControlState::APPLY:
        motor.set_constant_force(apply_force);
        apply_start_time = std::chrono::steady_clock::now();
        break;
    case ControlState::HOLD_CONTACT:
        hold_contact_start_time = std::chrono::steady_clock::now();
```

```
        break;
    case ControlState::AUTO_ZEROING:
        break;
    }
}

int main()
{
    motor.open_serial_port(serial_port);

    motor.set_mode(MotorMode::SleepMode);

    configureHapticEffects();

    motor.enable_haptic_effects(
        ORCAReg::HAPTIC_STATUS_Values::CONSTANT_Mask +
        ORCAReg::HAPTIC_STATUS_Values::SPRING_0_Mask +
        ORCAReg::HAPTIC_STATUS_Values::SPRING_1_Mask +
        ORCAReg::HAPTIC_STATUS_Values::SPRING_2_Mask +
        ORCAReg::HAPTIC_STATUS_Values::DAMPER_Mask
    );

    //Command the motor to enter haptic mode after autozeroing is completed
    motor.write_register_blocking(ZERO_MODE, ZERO_MODE_AUTO_ZERO_ENABLED);
    motor.write_register_blocking(AUTO_ZERO_EXIT_MODE, MotorMode::HapticMode);
    motor.set_mode(MotorMode::AutoZeroMode);

    enterState(ControlState::AUTO_ZEROING);

    while (1) {
       auto now = std::chrono::steady_clock::now();
       static auto elapsed_cycle_time = 0;
       static auto elapsed_hold_time = 0;

       detected_velocity =
motor.read_wide_register_blocking(ORCAReg::SHAFT_SPEED_MMPS).value;
       detected_force = motor.read_wide_register_blocking(ORCAReg::FORCE).value;
       motor_position = motor.get_position_um().value;
       switch (current_state) {
```

```
            case ControlState::AUTO_ZEROING:     //Wait for motor to complete
autozeroing
                if (motor.read_register_blocking(ORCAReg::MODE_OF_OPERATION).value
== MotorMode::HapticMode) {
                    enterState(ControlState::RETURN_HOME);
                }
            break;
            case ControlState::RETURN_HOME:      //Return to home position and wait
for next cycle trigger
                elapsed_cycle_time = ms(now - cycle_start_time);
                if (elapsed_cycle_time >= cycle_time_ms) {
                    enterState(ControlState::APPLY);
                }
                break;
            case ControlState::APPLY:            //Apply force and detect contact,
or an empty stroke, if timeout reached abandon
                if (detect_contact()) {
                    enterState(ControlState::HOLD_CONTACT);
                }
                else if (detect_empty()) {
                    enterState(ControlState::RETURN_HOME);
                }
                else if (ms(now - apply_start_time) > max_apply_time)
                {
                    enterState(ControlState::RETURN_HOME);
                }
                break;
            case ControlState::HOLD_CONTACT:     //Hold contact for specified time
                elapsed_hold_time = ms(now - hold_contact_start_time);
                if (elapsed_hold_time >= hold_contact_time_ms) {
                    enterState(ControlState::RETURN_HOME);
                }
                break;
        }
    }
     return 0;
}
```