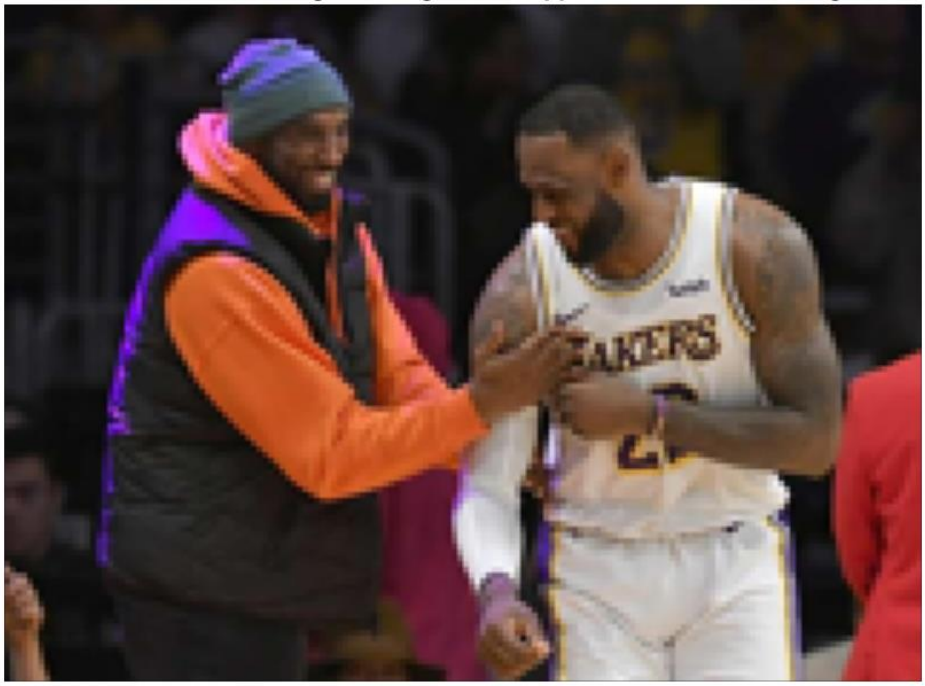
Gaussian and Average Moving Filters Applied on Retrieved Image



مشاهده می‌شود که کرنل‌های یاد شده، تضاد میان پیکسل‌های تصویر بازیابی‌شده را کاهش داده و تصویر به دست آمده کیفیت بهتری دارد.

در ادامه برنامه‌ای می‌نویسیم که لبه‌های یک کاغذ را از روی تصویر مشخص کند. الگوریتم این برنامه در ادامه آمده است:

تابع detect_page که ورودی‌های آن fileName، salt_pepper_ratio_H، salt_pepper_ratio_V، accuracyUp، accuracyDown، accuracyLeft، accuracyRight و precision هستند -که به ترتیب بیانگر نام فایل، نرخ نویز فلفل نمکی فیلتر Line H، نرخ نویز فلفل نمکی فیلتر Line V، آستانه دقت بخش بالای عکس، آستانه دقت بخش پایین عکس، آستانه دقت بخش چپ عکس، آستانه دقت بخش راست عکس و تعداد نقاط خروجی تابع detectHarrisFeatures هستند- را تعریف می‌کنیم.

۱. در ابتدای این تابع، عکس موردنظر را بارگذاری کرده و پس از سیاه سفید کردن آن، در ماتریس page ذخیره می‌کنیم. برای کار کردن با این عکس لازم است که درایه‌های آن از uint8 به عدد تبدیل شوند.

۲. ماتریس کرنل‌های line_H و line_V را تعریف می‌کنیم و با conv2 کرنل Line H را روی عکس اجرا می‌کنیم. سایز عکس را در متغیر page_size نگه می‌داریم.

۳. با پارامتر salt_pepper_ratio_H که از ورودی تابع خوانده شده -و به طور پیش‌فرض 5 درنظر می‌گیریم- و فیلتر median، نویز فلفل نمکی موجود در اطراف عکس را از بین می‌بریم. دقت شود که تغییر پارامتر یادشده، در کیفیت خروجی نهایی موثر است و می‌توان با آزمون و خطا بهترین مقدار آن را یافت. به علاوه، افزایش یا کاهش بیش از حد آن نیز موجب رخ دادن خطا در ادامه‌ی کار می‌شود.

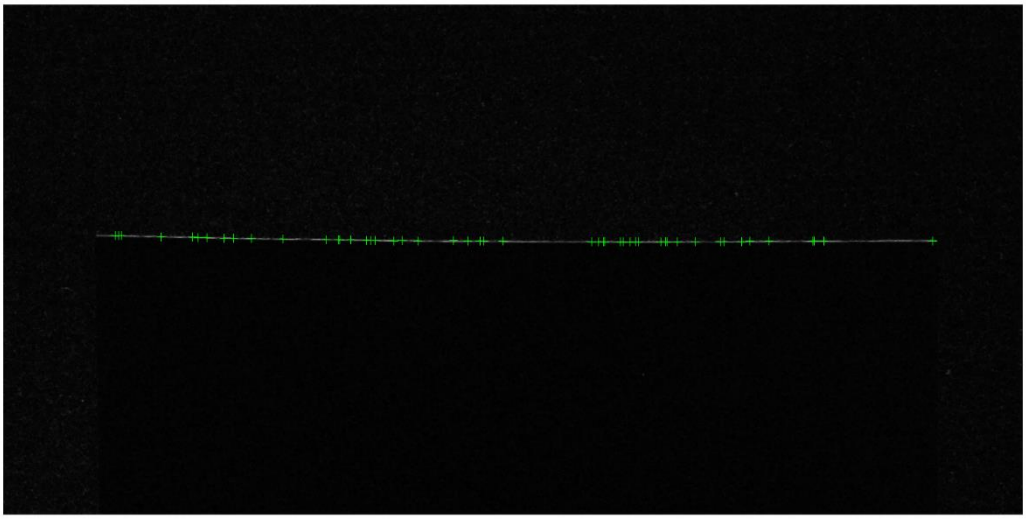
۴. عکس را به دو بخش pageUp و pageDown تقسیم می‌کنیم. این کار به بالا رفتن دقت فرایند تشخیص لبه‌های بالا و پایین کاغذ کمک می‌کند. بنابراین باید دقت شود که کاغذ موجود در عکس ورودی این الگوریتم در میان عکس باشد.

۵. این الگوریتم به کمک تابع detectHarrisFeatures متلب انجام می‌شود. این تابع در ابتدا pageUp را به عنوان ورودی گرفته و سپس، تعداد نقاط گوشه‌ای آن را خروجی می‌دهد. متغیر precision -که برابر ۵۰ فرض شده است-، تعداد نقاط خروجی این تابع را مشخص می‌کند. دقت شود که در صورتی که تصویر پس‌زمینه‌ی کاغذ موردنظر تیره نیست و تضاد کمی با رنگ خود کاغذ دارد، بالا بودن precision به تولید خطا منجر می‌شود. بنابراین برای تصاویری که شرایط یادشده را دارد باید precision کمی را مشخص کرد. در نهایت، نقاط خروجی این تابع را در متغیر cornersUp نگه می‌داریم. این متغیر یک شی دارای ۳ ویژگی Location، Metric و Count خواهد بود که ماتریس Location آن را در متغیر pointsUp نگه می‌داریم (همین مراحل به صورت موازی برای بخش پایین عکس نیز انجام می‌شود).

۶. از آنجا که کرنل Line H خطوط افقی تصویر را مشخص می‌کند و نقاط روی این خطوط توسط تابع detectHarrisFeatures شناسایی می‌شوند، برای یافتن معادله خط گذرنده از این نقاط

کافیست عرض آن‌ها را در Yup (یا YDown) نگه داشته و با محاسبه مربعات تفاضل دو به دوی عناصر Yup (یا YDown) و درنظر گرفتن آستانه دقت بخش بالای عکس (یا آستانه دقت بخش پایین عکس) -که در صورتی که 1- باشد، به طور پیش‌فرض برابر با میانگین عناصر Yup (یا YDown) درنظر گرفته می‌شود- **نقاطی را که روی خط افقی نیستند**، از cornersUp (یا cornersDown) حذف کنیم. در این مرحله، تصویر نقاط نهایی بخش بالای کاغذ را نمایش می‌دهیم.

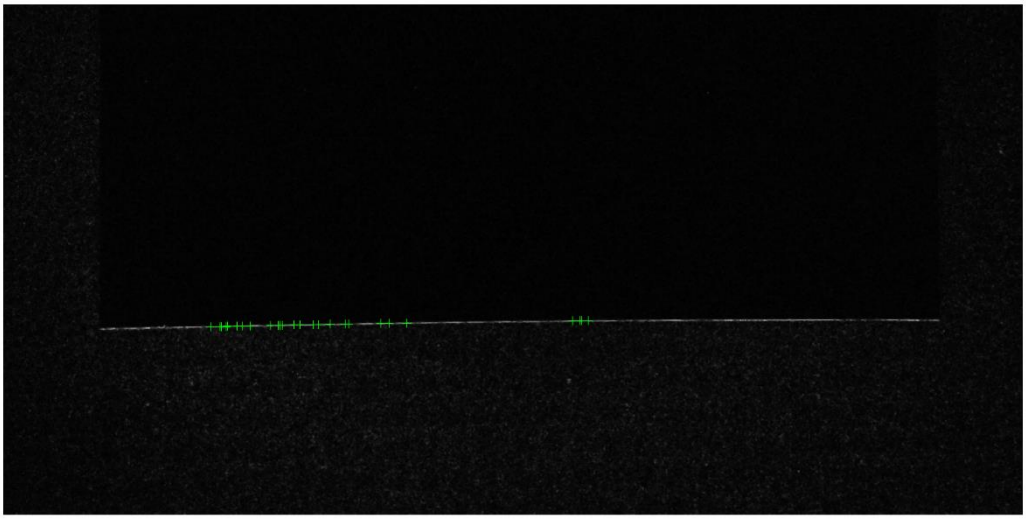
* عکس مورد آزمایش در این مراحل، page.jpg بوده و پارامترهای دیگر تابع detect_page مقادیر پیش‌فرض خود را خواهند داشت.



نقاط خط افقی بخش بالای عکس

۷. در این مرحله با استفاده از نقاط cornersUp.Location و تابع fit، معادله خط گذرنده از آن‌ها -یعنی lineup- را می‌یابیم.

۸. مراحل فوق را برای بخش پایین تصویر نیز انجام داده و تصویر تولید شده را مشاهده می‌کنیم.



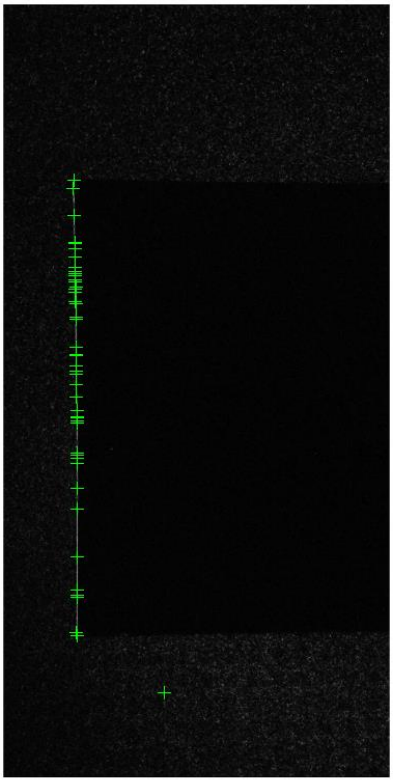
نقاط خط افقی بخش پایین عکس

۹. با conv2 کرنل Line V را روی عکس اجرا می‌کنیم. سایز عکس را در متغیر page_size نگه می‌داریم.

۱۰. با پارامتر salt_pepper_ratio_V که از ورودی تابع خوانده شده -و به طور پیش‌فرض 5 درنظر می‌گیریم- و فیلتر median، نویز فلفل نمکی موجود در اطراف عکس را از بین می‌بریم.

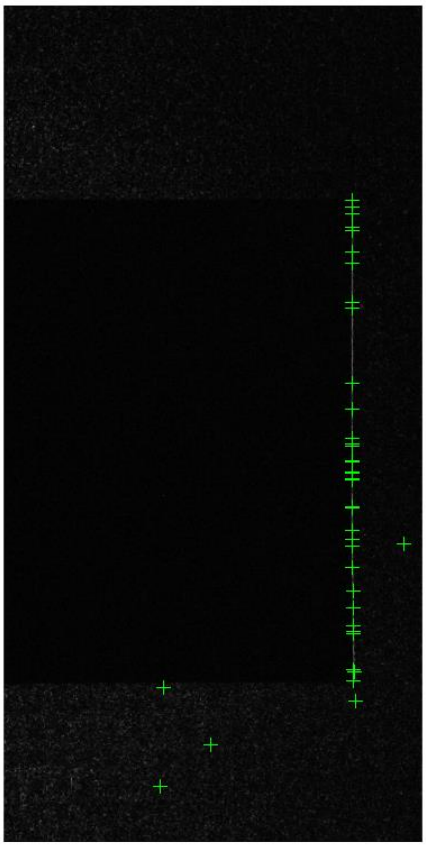
۱۱. عکس را به دو بخش pageLeft و pageRight تقسیم می‌کنیم. این کار به بالا رفتن دقت فرایند تشخیص لبه‌های سمت چپ و راست کاغذ کمک می‌کند. بنابراین باید دقت شود که کاغذ موجود در عکس ورودی این الگوریتم در میان عکس باشد.

۱۲. به همان روشی که پیش‌تر ذکر شد، نقاط روی خطوط عمودی بخش چپ و راست تصویر را یافته و معادله خط گذرنده از آن‌ها را به ترتیب در lineLeft و lineRight ذخیره می‌کنیم و تصاویر تولیدشده را مشاهده می‌کنیم.



نقاط خط عمودی بخش سمت چپ عکس

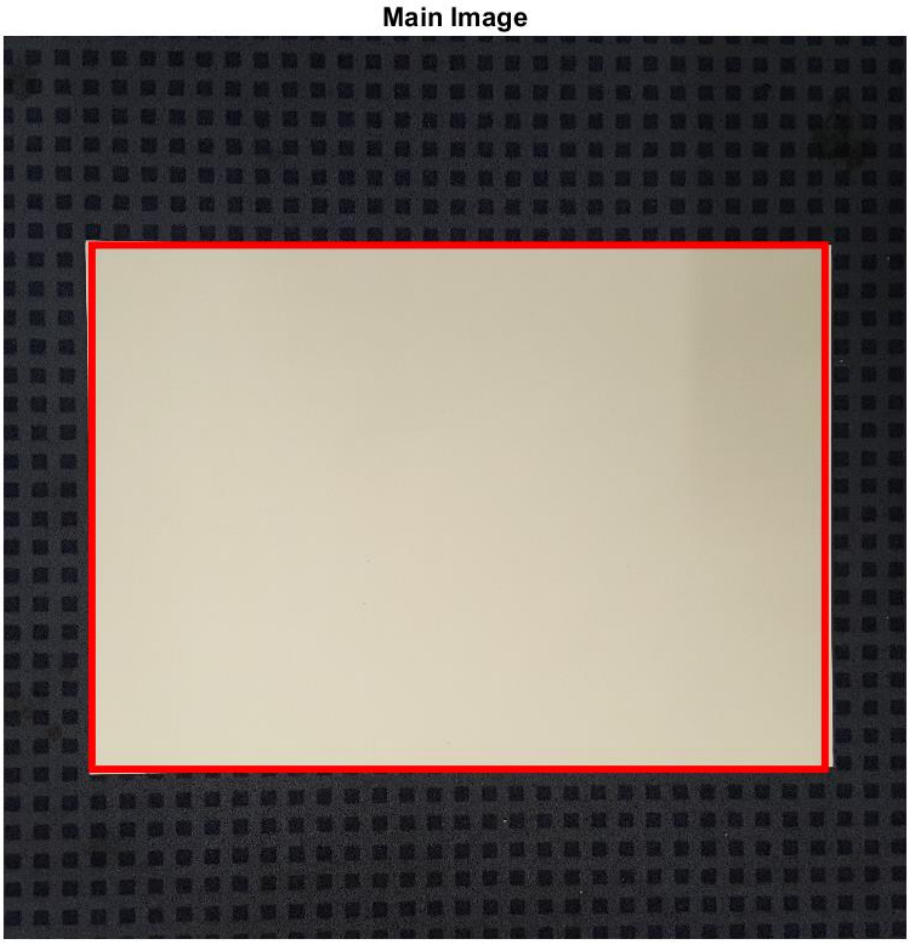
مشاهده می‌شود که در میان نقاط نهایی، نقطه‌ای وجود دارد که روی خط عمودی نیست و حضور آن باعث انحراف خط lineLeft و در نهایت، کاهش کیفیت خروجی می‌شود. اما می‌توان با تنظیم درست پارامترهای تابع detect_page -که در این آزمایش مقادیر پیش‌فرض خود را داشتند- نقاطی مانند این نقطه را حذف کرد و کیفیت خروجی را بالا برد. در ادامه، پارامترهایی که نتیجه بهتری برای این عکس می‌دهند ارائه خواهد شد.



نقاط خط عمودی بخش سمت راست عکس

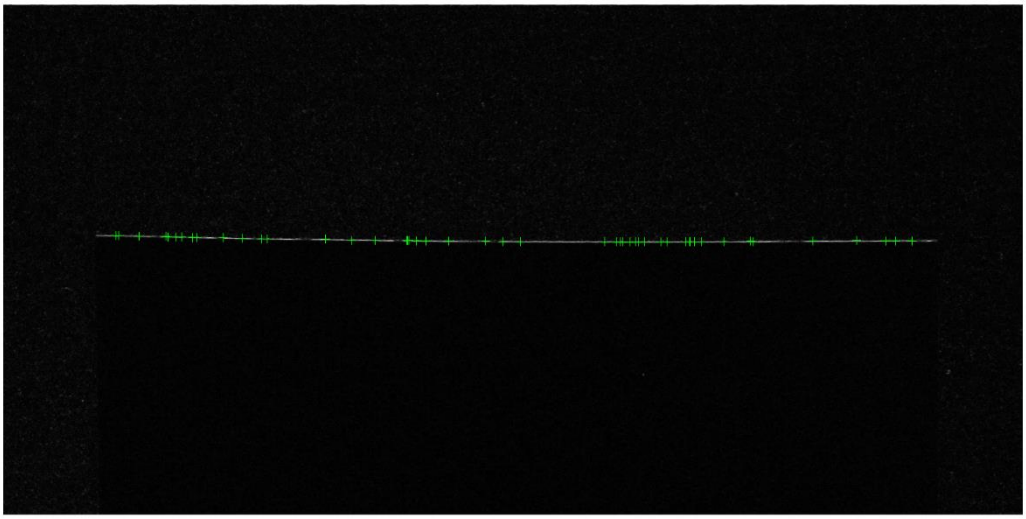
مشاهده می‌شود که در این تصویر نیز، در میان نقاط نهایی، چهار نقطه وجود دارد که روی خط عمودی نیستند و حضور آن‌ها باعث انحراف خط lineRight و در نهایت، کاهش کیفیت خروجی می‌شود.

۱۳. برای بهبود معادله خطوط حاصل‌شده در بخش‌های پیش، هنگام محاسبه محل تلاقی آن‌ها برای کشیدن مستطیل قرمز روی تصویر اصلی، از **میانگین عرض‌ها و طول‌های محل‌های تلاقی** استفاده می‌شود تا انحراف خطوط مستطیل کمینه شود. در نهایت با تابع rectangle، تصویر نهایی حاصل می‌شود.

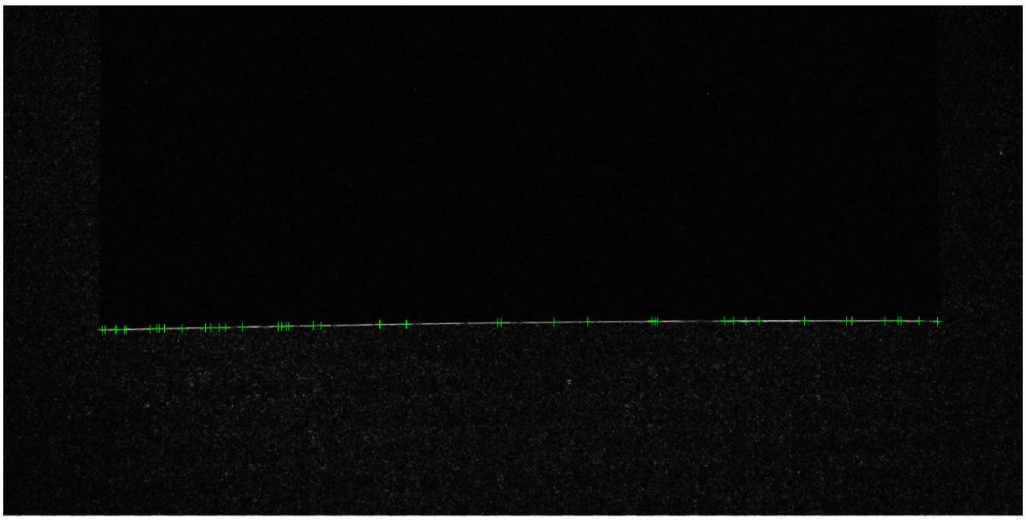


تصویر نهایی. با پارامترهای پیش‌فرض تابع detect_page

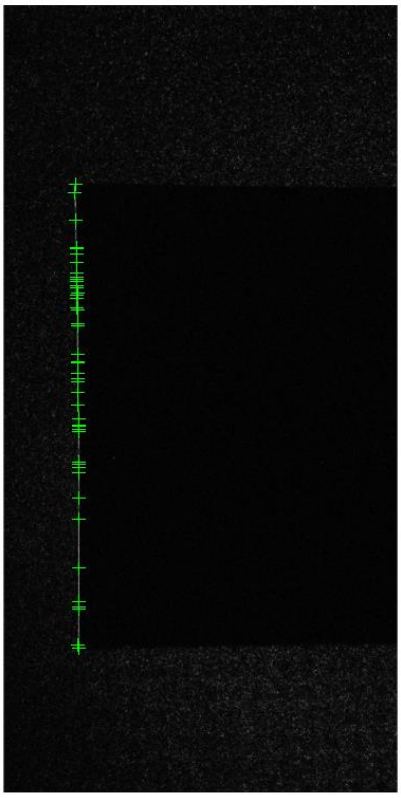
❋ حالا با کمی آزمون و خطا پارامترهایی را که برای تصویر page.jpg بهترین نتیجه را به دست می‌دهند (به ترتیب 3، 5، 0.1، 0.14، 0.1 و 50)، یافته و تصاویر نهایی هر مرحله را مشاهده می‌کنیم:



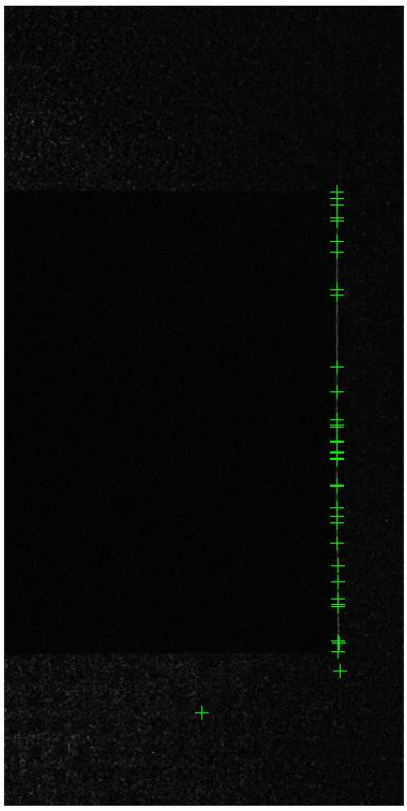
نقاط خط افقی بخش بالای عکس. با پارامترهای بهینه برای تابع detect_page



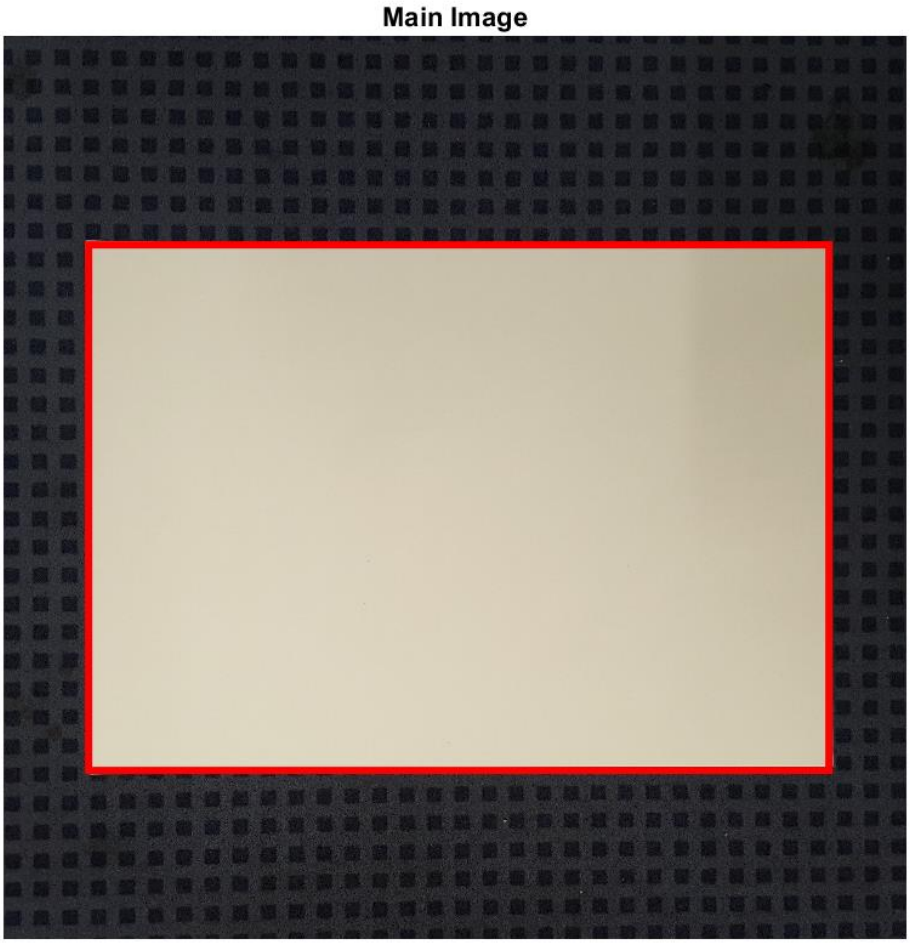
نقاط خط افقی بخش پایین عکس. با پارامترهای بهینه برای تابع detect_page



نقاط خط عمودی بخش سمت چپ عکس. با پارامترهای بهینه برای تابع detect_page



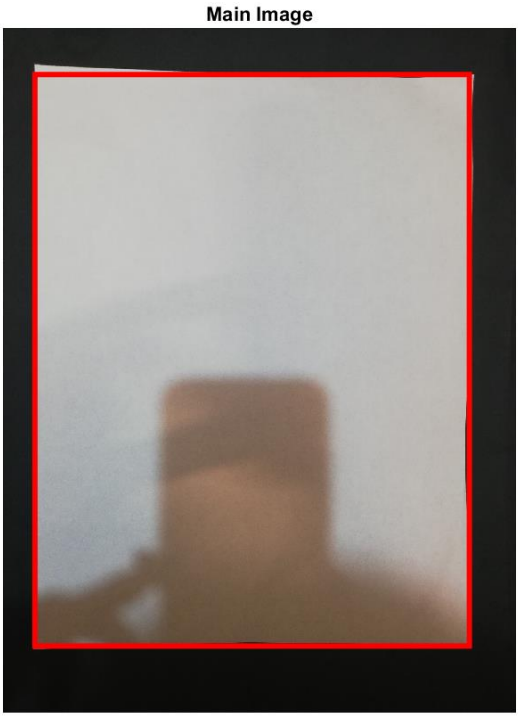
نقاط خط عمودی بخش سمت راست عکس. با پارامترهای بهینه برای تابع detect_page



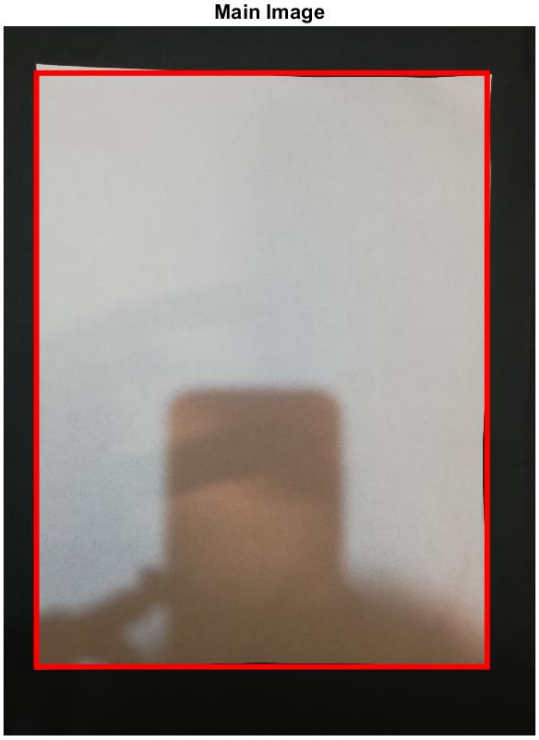
تصویر نهایی. با پارامترهای بهینه برای تابع detect_page

مشاهده می‌شود که در صورت یافتن پارامترهای بهینه، نتیجه بسیار بهتر خواهد بود.

همین مراحل بر روی عکسی با نام p.jpg نیز تکرار شده که ابتدا نتیجه را با درنظر گرفتن پارامترهای پیش‌فرض و سپس با درنظر گرفتن پارامترهای بهینه (به ترتیب 3، 3، 0.1، 0.098، 0.1، 0.3 و 50) مشاهده می‌کنیم:



نتیجه. با پارامترهای پیش‌فرض تابع detect_page



نتیجه. با پارامترهای بهینه برای تابع detect_page